Estudios de Teoría Literaria
Revista digital
Artes
Letras
Humanidades
29
año 12
noviembre 2023
ISSN 2313-9676

Del Vecchio, Alejandro. "De Góngora a JavaScript: lectura de un poema desde los estudios críticos del código"
Estudios de Teoría Literaria. Revista digital: artes, letras y humanidades, noviembre de 2023, vol. 12, nº 29, pp. 42-53.

De Góngora a JavaScript: lectura de un poema desde los estudios críticos del código

From Góngora to JavaScript: reading a poem from Critical Code Studies

Alejandro Del Vecchio[1]

ORCID: 0000-0002-4179-8595

Recibido: 14/08/2023 ‖ Aprobado: 02/10/2023 ‖ Publicado: 17/11/2023

Resumen
El presente artículo ensaya una lectura de sendas versiones de "Ars poetica", code poem de Alejandro Corredor Parra, a partir de los aportes de los denominados "Estudios críticos del código". La era de la revolución electrónica promueve la emergencia de prácticas de escritura literaria experimentales, cuya lectura exige profundizar la alfabetización digital y la creación de marcos teóricos apropiados, para los que los ECC constituyen un aporte ineludible. Si bien el texto de Corredor Parra aparece originalmente publicado en papel, se trata de un programa funcional, hecho que sacude el estatuto de la pieza artística, al desdoblarla en dos dimensiones: una vinculada con su materialidad significante (en tanto texto literario) y otra con su potencialidad performativa en un dispositivo digital (en tanto código informático).

Palabras clave
Estudios críticos del código; poema de código; Alejandro Corredor Parra; Arte poética

Abstract
This article proposes a reading of two versions of "Ars poetica", a code poem by Alejandro Corredor Parra, based on the contributions of the so-called "Critical Code Studies". The era of the electronic revolution promotes the emergence of experimental literary writing practices, the reading of which requires deepening digital literacy and the creation of appropriate theoretical frameworks, for which CCS constitute an unavoidable contribution. Although Corredor Parra's text appears originally published on paper, it is a functional program, a fact that shakes the status of the artistic piece, by unfolding it into two dimensions: one linked to its significant materiality (as a literary text) and another with its performative potential in a digital device (as computer code).

Keywords
Critical Code Studies; code poem; Alejandro Corredor Parra; poetic art

[1] Magister en Letras Hispánicas. Ayudante graduado en la cátedra "Literatura y cultura latinoamericanas 2" y miembro del grupo de investigación "Literatura y Cultura Latinoamericanas" dirigido por la Dra. Mónica Marinone (Universidad Nacional de Mar del Plata-CELEHIS). Profesor asociado en las cátedras de "Gramática castellana 1" y "Gramática castellana 2" (Universidad CAECE).

Code isn't just functional, it's poetic.
Jonathan Keats

article.Init()
{

Consideramos el siguiente programa diseñado en *Processing*[2], firmado por el artista colombiano Alejandro Corredor Parra[3] y aparecido en *code {poems}* (2012), volumen compilado por Ishac Bertran:

ARS POETICA

```
String silence="                    ";
String idea="This is'nt pØetry.";
String draft;
String[]poem=new String[idea.length()];
void setup(){
 draft=idea;
 Write();
 ReThink();
}
void draw(){
ReWrite();
}
void Write(){
println (draft);
}
void ReThink(){
for(int decomp=0;decomp<draft.length();decomp++)
{poem[decomp]=draft.substring(decomp,decomp+1);}
}
void ReWrite(){
 byte seek=byte(random(0, poem.length));
 poem[seek]=" ";
 String poetry=join(poem,"");
 println(poetry);
 if(poetry.equals(silence)){
 println("."); noLoop();}
}
```

Frente a la singularidad del objeto, no es improbable que tanto el horizonte de expectativas del programador como el del lector de literatura resulten defraudados.

El programador acaso encuentre extraña la presencia de un título (por si fuera poco en latín) para el código, el que a su vez no aparece incluido (como podría esperarse) en una compilación de algoritmos o en un manual de programación, sino en un libro de "*code poems*" o "poemas de código"[4]. Percibirá como singular, además, el significado connotativo del programa y probablemente lo deseche por inútil en términos pragmáticos.

Para el potencial lector de poesía, las estructuras del texto, sus llaves, sus operadores lógicos y matemáticos, sus sintagmas y sintaxis anómalos (entre otros factores) resultarán sin duda disruptivos. No obstante, el lector versado en teorías literarias formalistas acaso distinga

[2] Ver https://processing.org/overview.

[3] Alejandro Corredor Parra (a.k.a El Sr. [Aleph]) es un escritor de poesía y literatura digital nacido en Colombia. Su blog: https://corredorparra.com/

[4] Los editores abrieron una convocatoria entre el 22 de febrero y el 31 de mayo de 2012. Las reglas para enviar propuestas eran simples: (1) el poema no podía exceder los 0,5 KB, y (2) el poema tenía que compilar sin errores. Se recibieron 190 poemas de 30 países diferentes.

las líneas de código como versos (no solo por su disposición espacial sino por sus anáforas) y, sobre todo, no desconozca el extrañamiento provocado por el discurso poético propio de, por ejemplo, las vanguardias históricas. A su vez, el título, operador fundamental de sentido, lo conectará de modo inmediato con la tradición literaria. Y por último, desde teorías de corte socioinstitucional, nuestro lector modelo advertirá que el texto circula en una compilación de poemas, en un contexto en el que las fronteras que circunscriben la poesía (y la literatura en general) se han vuelto más permeables.

Pero para complejizar un poco más nuestro objeto de estudio, "Ars poetica" podría ejecutarse en un ambiente de desarrollo de código abierto. Es decir, se trata de un programa perfectamente funcional, diseñado (como anticipé) para correr mediante la librería Processing, hecho que, como mínimo, sacude el estatuto de la pieza artística, al desdoblarla en dos dimensiones: una vinculada con su materialidad significante (en tanto texto literario) y otra con su potencialidad performativa en un dispositivo digital (en tanto código informático). Como advierte Rita Raley, aunque tanto el código como el texto pueden divertir, asombrar, informar y deleitar, ambos pueden ser escritos y leídos, ambos son performantes y pueden operar cambios en el mundo, uno puede ser ejecutado por la computadora y el otro no (9).

Queda claro, por ende, que solo constituye código aquella escritura que puede compilar o ejecutarse (ya que respeta estrictamente la sintaxis de un lenguaje formal determinado) más allá del soporte en el que circule. Cuando se utiliza un pseudocódigo, en cambio, en el que se hibrida lengua natural y lenguaje de programación, no podría entonces hablarse sino únicamente de texto. Belén Gache define esta última clase de *codework*[5] como aquella que "se caracteriza por moverse en una zona de contaminación y contagio entre la dimensión del código y el texto literario, entendiendo la escritura igualmente como código y definiendo tanto a una como a otro como conjuntos de signos alfabéticos" (199).

Un ejemplo de este segundo tipo de *codework* es el poema "Shock" de Belén García Nieto:

```
switch (descontento) {
       case 'leve':
              system.apply('TV');
       case 'moderado':
              system.apply('Policía');
       case 'severo':
              system.apply('Fascismo');
       default:
              system.apply('Shock');
}
```

En muchos lenguajes de programación, un *switch case* o *switch statement* constituye un mecanismo para permitir que el valor de una variable o expresión cambie el flujo de control de la ejecución del programa. Pero "Shock", tal como se presenta al lector (y a diferencia del "Ars poetica" de Corredor Parra) no podría ejecutarse, ya que su compilación arrojaría como mínimo dos errores: la indefinición de la variable *descontento* y de la función *system.apply*(). Su comprensión, empero, constituye un escollo salvable para el lector bilingüe, más allá de que posea o no saberes específicos. El poema de García Nieto, por otra parte, evidencia el problema de la traducción en este tipo de textos que articulan "zonas de contaminación y contagio": términos como *switch, case, default* no podrían, en principio, aparecer en español, como sucede con el nombre de la variable *descontento*, los valores evaluados en el *case* y los

[5] Alan Sondheim acuña el término *codework* en 2001 para denominar prácticas de escritura en las que el código emerge, se hace visible y se entreteje con el lenguaje natural.

parámetros de *system.apply*, clase cuyo nombre polisémico (sistema informático, sistema político) confunden (o impugnan) el adentro y el afuera del texto como en una cinta de Moebius.

Por otra parte, si bien "código" y "texto" poseen –como vemos– singularidades distintivas, también pueden pensarse como equivalentes (al menos en una dirección), ya que el código fuente ejecutable también es un texto, un sistema de signos con sintaxis y retórica específicas, producido en un contexto determinado, cuyo potencial significante excede su dimensión funcional.

En efecto, producto de la revolución digital y de la emergencia de culturas de la programación, el código fuente de los lenguajes informáticos –normalmente escatimado al ojo humano y confinado a una caja negra– deviene elemento estético para la producción artística y literaria, motor del activismo digital y –nos importa sobre todo– objeto de investigación de los llamados "estudios críticos del código", que abordan los diversos modos en los que el código (no) circula, (no) pone en escena su materialidad y (no) prevé su recepción, entre otros aspectos.

```
}

article.Setup()
{
```

Para esta línea de trabajo, *Critical Code Studies* (2020) de Mark Marino constituye un texto fundacional, en tanto instaura las bases para consolidar el campo disciplinar, emergente en la era digital, al que acabo de aludir, los "estudios críticos del código" (ECC), que el propio Mark Marino define como:

> an approach to code studies that applies critical hermeneutics to the interpretation of computer code, program architecture, and documentation within a sociohistorical context. CCS holds that the lines of code of a program are not value-neutral and can be analyzed using the theoretical approaches applied to other semiotic systems, in addition to particular interpretive methods developed specifically for the discussions of programs (39).
>
> [una aproximación a los estudios de código que aplica la hermenéutica crítica a la interpretación del código informático, la arquitectura del software y su documentación, en un contexto sociohistórico. Los ECC sostienen que las líneas de código de un programa no son ideológicamente neutrales y pueden ser analizadas utilizando enfoques teóricos aplicados a otros sistemas semióticos, además de métodos interpretativos particulares desarrollados específicamente para el análisis de programas.]

El objeto principal de indagación de los ECC, el código fuente (ya sea que permanezca oculto en los dispositivos, ya sea que emerja a la superficie de una interfaz)[6], es concebido como "reino semiótico" distintivo, susceptible de ser leído y analizado como texto de cultura y, por consiguiente, considerado también en su dimensión estética y pragmática.

En este sentido, en *Beautiful Code* (2007), editado por Andy Oram y Greg Wilson, Wilson afirma: "I saw that programs could be more than just instructions for computers. They

[6] Por ejemplo en producciones como *Memorias de Ariadna Alfil* (2016) o *Amazon* (2019) de Eugenio Tisselli, poeta, *net.artista* e ingeniero informático nacido en México.

could be as elegant as well-made kitchen cabinets, as graceful as a suspension bridge, or as eloquent as one of George Orwell's essays" [Vi que los programas podían ser más que simples instrucciones para computadoras. Pueden ser tan elegantes como gabinetes de cocina bien hechos, tan gráciles como un puente colgante o tan elocuentes como un ensayo de George Orwell] (XVI). El volumen referido compendia trabajos de desarrolladores de software que encuentran belleza en los pequeños detalles de un programa diseñado con estilo (en su estructura o en las técnicas utilizadas para construirlo). Jon Bentley, autor de un pionero algoritmo para ordenar elementos de un vector (*Quicksort*), lo sintetiza en su ensayo "The Most Beautiful Code I Never Wrote" ["El código más hermoso que jamás haya escrito"]:

> I once heard a master programmer praised with the phrase, «He adds function by deleting code». Antoine de Saint-Exupéry, the French writer and aviator, expressed this sentiment more generally when he said, «A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away». In software, the most beautiful code, the most beautiful functions, and the most beautiful programs are sometimes not there at all (29).
>
> [Una vez escuché a un maestro programador elogiar la frase: "Él agrega funcionalidad eliminando código". Antoine de Saint-Exupéry, el escritor y aviador francés, expresó este sentir de manera más general cuando dijo: "Un diseñador sabe que ha alcanzado la perfección no cuando no queda nada por agregar, sino cuando no queda nada por quitar". En el software, el código más bonito, las funciones más bonitas y los programas más bonitos a veces no están allí.]

Depurar un programa como se depura un poema / depurar un poema como se depura un programa. Aunque por medio de un acervo de herramientas lingüísticas, lógicas y matemáticas más limitadas, el programador juega con la materialidad del código como el poeta juega con la materialidad de la lengua. Por eso Jamie Allen destaca que, a pesar de substanciales restricciones formales y estandarizaciones intrínsecas, el estilo de programación de un desarrollador de software siempre se visibiliza en el código, ya sea en sus estrategias para optimizar el fuente que ejecuta la computadora, ya sea a través de su organización visual y de comentarios para otros codificadores. Como ocurre con la lengua poética, el código no es nunca meramente denotativo o funcional, sino una condensación escrita del ánimo, personalidad y cosmovisión de su autor, además de insertarse en un contexto cultural y político en el que las arquitecturas de software y hardware evolucionan (Bertran vi).

Si, asimismo, el software aparece entretejido en la vida contemporánea –económica, cultural, creativa, políticamente– "de maneras tanto obvias como casi invisibles" (Marino ix) el tono inquietante del adjetivo "invisibles" evoca (a la vez que reaviva) la sospecha sobre los códigos ocultos en dispositivos y plataformas, y los intereses a los que responden. Si "gobierno" y "cibernética" albergan una etimología común, las políticas gubernamentales en la era de la episteme de la información buscan "conducir las conductas".

Sin acceder al código, por lo tanto, el trabajo del analista queda restringido a valorar solo los efectos de las operaciones aparentes que un programa realiza. "Crear el software o ser el software" / "programar o ser programado" son las fórmulas que elige Douglas Rushkoff para advertirlo en otros términos. O –como señala Bifo Berardi en el prefacio a *Speaking Code: Coding as Aesthetic and Political Expression* (2012)– si podemos decir que el código nos habla (impregnando y formateando nuestras acciones), también "hablamos código" de diversas maneras, ya que el poder está cada vez más inscripto en el código (Cox y McLean 7).

En esta línea de pensamiento, ya en 2006 Marino proclamaba la necesidad de sacar el código de su caja negra para explorar el modo en que su complejo sistema semiótico produce significación, en tanto texto, mientras circula en contextos diversos:

> As a new media scholar trained in literary theory, I would like to propose that we no longer speak of the code as a text in metaphorical terms, but that we begin to analyze and explicate code as a text, as a sign system with its own rhetoric, as semiotic communication that possesses significance in excess of its functional utility (Marino 39).
>
> [Como estudioso de los nuevos medios formado en teoría literaria, me gustaría proponer que ya no hablemos del código como texto en términos metafóricos, sino que comencemos a analizar y explicar el código como texto, como sistema de signos con su propia retórica, como comunicación semiótica que posee significado que excede su utilidad funcional.]

Las implicancias teóricas de este enfoque para redefinir categorías clásicas como autor, lector, escritura, lectura (entre otras) son acaso inconmensurables. No obstante, como anticipé, en tanto "code is at once what it is and what it does" (51) [el código es a la vez lo que es y lo que hace], leerlo *barthesianamente* como texto no implica negar su especificidad maquínica originaria. En cualquier caso, el significado del código está marcado por la tensión entre ambigüedad (porque es social y polisémico) y a la vez precisión (porque es tecnológico y su sintaxis estricta).

En el apartado "How to Interpret Code" [Cómo interpretar código] del volumen aludido, Marino propone incluso una extensa lista preliminar –y por ende abierta– de técnicas y enfoques probadamente eficaces y fructíferos para la interpretación de código. Un paso ineludible consiste en desarrollar una lectura y relectura comprensivas de su funcionamiento: el crítico necesita determinar la función del código antes de examinar su significado extrafuncional. Otro punto central reside en la investigación de su contexto de producción: detrás de cualquier código late la historia de su creación, las ideas detrás de su concepción, las coordenadas de espacio y tiempo de su desarrollo, opciones de diseño, métodos y nombres de variables, e incluso deficiencias en su escritura. Asimismo, otro enfoque valioso consiste en investigar la evolución del código fuente (como el filólogo investiga manuscritos) para indagar diacrónicamente versiones y ampliaciones, como estrategia útil para reconstruir una suerte de biografía del código. También investigar sus componentes y estilo resulta fundamental: "Code is never simply code" (235) [El código nunca es simplemente código], dice Marino. Tiene cualidades: claridad frente a ofuscación, verbosidad frente a concisión, elegancia frente a negligencia. Otras exploraciones recientes se centran en la ética incrustada en el código (por ejemplo, el potencial de sesgo destructivo en aquel que subyace a la inteligencia artificial). De igual modo, el código a menudo va acompañado de escritura adicional, es decir, de un sistema paratextual: ya sea documentación anexa, comentarios de los desarrolladores o comunicaciones entre desarrolladores. Y como en cualquier buen análisis textual, cuanto mejor los estudiosos comprendan el lenguaje –enraizado en historias intelectuales y paradigmas– y sus orígenes, mejor comprenderán el código; como en cualquier análisis semiótico, los signos tienen significado en relación con el lenguaje al que pertenecen. Más allá de estos enfoques, Marino señala que:

> Code itself is neither the end nor the beginning of this reflection, but as an expression of thought, as a trace record of labor and development history, as an artistic medium, and as a connection point in human-machine assemblages, code offers an opportunity

> to reflect on technoculture with symbols that are at once completely unambiguous and at the same time open to interpretation (239).
>
> [El código en sí mismo no es ni el principio ni el final de estas reflexiones, pero como expresión del pensamiento, como registro de la historia del trabajo y el desarrollo, como medio artístico y punto de conexión en los ensamblajes humano-máquina, el código ofrece una oportunidad para reflexionar sobre la tecnocultura, a través de símbolos que son completamente inequívocos y a la vez están abiertos a interpretación.]

En este contexto, el volumen *Critical Code Studies* constituye, por lo tanto, el grado cero en la teorización de una práctica, tensionada entre la lectura contextual y el *close-reading*, que se orienta a analizar fenómenos culturales a través del código y su sistema paratextual, por lo que resulta idónea para abordar objetos complejos como el "Ars poetica" de Corredor Parra. El adjetivo "críticos", por su parte, inscribe su arco conceptual en una genealogía relacionada con teorías competentes para deconstruir mecanismos opresivos, desde el feminismo al marxismo, y desde los estudios *queer* a los poscoloniales, entre otras vertientes. Por eso, los grupos de trabajo interdisciplinarios de ECC[7] se basan en lecturas colectivas del código. El esfuerzo colaborativo articulado desde muchas miradas (tamizadas por lentes diversas) y muchas perspectivas resulta indispensable para desentrañar significados profundos.

```
}

article.Run()
{
```

A partir de los planteos de los ECC, ensayemos entonces una lectura posible de "Ars poetica", texto cuyo título remite no solo a la tradición clásica (desde Platón y Aristóteles hasta la *Epístola ad Pisones* de Horacio), sino a toda una serie de escritores latinoamericanos que han escrito sus "Arte poética"[8]. Según Corredor Parra, "este trabajo consiste en una transcreación inspirada en el poema «Hasta que el verso quede»[9] de Francisco Hernández en la que el silencio es el único resultado posible de la tarea de depuración verbal que es en sí misma poética". Si los primeros versos de Hernández proponen "Quitar la carne, toda, / hasta que el verso quede / con la sonora oscuridad del hueso", la pieza del colombiano escenifica la descomposición en el nivel de los grafemas de un único verso hasta que solo queda silencio[10].

[7] Visitar https://criticalcodestudies.com/ccswg.html

[8] Vicente Huidobro, Pablo Neruda, Alfonso Reyes, Jorge Luis Borges, Manuel Bandeira, por citar solo algunos.

[9] Quitar la carne, toda, / hasta que el verso quede / con la sonora oscuridad del hueso. / Y al hueso desbastarlo, pulirlo, aguzarlo / hasta que se convierta en aguja tan fina, / que atraviese la lengua sin dolencia / aunque la sangre obstruya la garganta.

[10] El tópico del silencio tiene un antecedente poderoso en el concretismo. En el poema de Eugen Gomringer "schweigen/silence", por ejemplo, el "silencio" se expresa mediante la ausencia de la palabra, es decir, con un recurso paralingüístico:

silencio silencio silencio
silencio silencio silencio
silencio silencio
silencio silencio silencio
silencio silencio silencio

La rigurosa geometrización simétrica del poema constituye, precisamente, un rasgo central de la poesía concreta.

Confinado al papel, el poema de Corredor Parra podría leerse linealmente, como un texto más, prescindiendo de su lógica y del efecto concreto de cada sentencia. Sin embargo, en tanto código ejecutable, incita a una lectura especializada. Es decir, la lectura de un programador de tecnologías web, capaz de entender su método de funcionamiento, a partir de operaciones mentales.[11] Así por ejemplo, el conocimiento de lenguajes de programación resulta imprescindible para decodificar el sentido pleno de la secuencia inicial de instrucciones:

```
String silence="                    ";
String idea="This is'nt pØetry.";
String draft;
String[]poem=new String[idea.length()];
```

La primera línea del poema define una cadena de caracteres (*String*), en este caso compuesta por espacios blancos. Si bien *String* es una etiqueta propia de la sintaxis de este y muchos otros lenguajes informáticos, el nombre de la variable, *silence* (más allá de estar en inglés por preferencia de Corredor Parra[12]), podría ser cualquier otro (e incluso, por supuesto, traducirse al español como "silencio", rasgo lingüístico presente en el texto citado de García Nieto). En este sentido, la elección de un término denotativo (y no de una abreviatura o simplemente una letra) apunta a afirmar la legibilidad del código y configurar un campo semántico que irá dotando de unidad significante a todo el texto: silencio, idea, borrador, poema, poesía, escribir, reformular, reescribir.

La segunda línea es neurálgica, ya que define la cadena *idea* (también transparente en español) mediante una oración, "This is'nt p0etry", en torno a la cual opera la pieza. A pesar de su aparente simplicidad, la frase –en cuya grafía irrumpe el alfabeto binario– exige considerar su polisemia y sus múltiples connotaciones culturales. La más evidente radica en la reescritura de la inscripción de la famosa colección de pinturas *La trahison des images* (1928-1929) de René Magritte: *Ceci n'est pas une pipe* [*This isn't a pipe* / *Esto no es una pipa*]. En *Les deux mystères* (1966) (figura 1), pintura que, varias décadas después, cierra la serie, Magritte pone en abismo la relación imagen-realidad explorada en las obras de la década del 20:

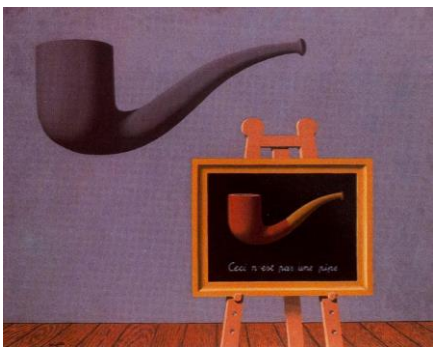

Figura 1

[11] Del mismo modo que el músico puede imaginar cómo suena una partitura, el programador puede imaginar (o recrear en papel mediante una "prueba de escritorio") el resultado de la ejecución de un código.

[12] En *Critical Code Studies*, Marino dedica el capítulo "FLOW-MATIC" a examinar proyectos artísticos que –a partir de la tensión entre código fuente y lenguajes naturales– proponen alternativas al predominio de herramientas de programación basadas en inglés, como forma de contrarrestar sus efectos imperialistas y coloniales sobre quienes programan.

El contraste entre las dos pipas parece sugerir que la de la pintura interior es una representación o símbolo y la otra, la que flota y carece del texto que niega su esencia, una pipa "real". Sin embargo, resulta evidente que ambas pipas yacen en el mismo lienzo, hecho que se advierte al contemplar el cuadro expuesto en una sala: no hay una pipa más "real" que otra, a pesar de lo que el acápite afirma.

La definición de este procedimiento retórico que, simplificando, supone la "obra dentro de la obra", exige precisiones, ya que existen diversas modalidades y matices en la descripción de esta estructura y, sobre todo, en los efectos de sentido propiciados. Según Lucien Dällenbach, existe precisamente un tipo de *mise en abyme* de naturaleza metatextual, a la que también denomina estructura abismada del código:

> […] la *mise en abyme* metatextual […] resulta más o menos evidente según sea el grado de homología entre código y enunciado-referente […] nada nos impide […] extender su alcance a algún *(a)* arte poético, algún *(b)* debate estético, algún *(c)* manifiesto, algún *(d)* credo, o alguna *(e)* indicación sobre la finalidad que el productor asigna a la obra o que la obra a sí misma se asigna –a condición de que tal arte poético, tales consideraciones estéticas, tal manifiesto, tal credo o tal marca de destino, sean […] asumidos por el texto de manera lo suficientemente visible como para que el reflejo metatextual pueda operar a guisa de *instrucciones de uso*, aprestando la lectura para que cumpla su tarea con menos dificultad: rehacer, como en un espejo, lo que su revés simétrico hizo antes que él; tomar la obra por lo que desea ser tomada (121-122).

Se trata de una figura que suele revelar algún principio compositivo del texto o pieza artística en los que se inserta o, en el caso de Magritte, de aspectos que rigen una poética. La estructura abismada opera como un modelo reducido (y repetido) que funciona como paráfrasis alegórica de los fundamentos de un programa estético. Pero si en Magritte la paradoja metapoética afinca en el contrapunto (o mejor contradicción) entre imágenes e inscripción, el cuestionamiento de la representación en el poema de Corredor publicado en papel en *code {poems}* se manifiesta en un texto/código que solo al ser ejecutado tendrá como rasgo singular su disolución plástica en tiempo real.

En su segunda línea, el error ortográfico en la contracción verbal (*is'nt*) –involuntario o no– carga de ambigüedad el enunciado. Detectado el equívoco o errata, puede leerse, en efecto, la negación del verbo "ser" ("Esto no es poesía"). De modo literal (y excluido el apóstrofe), en cambio, "nt" puede leerse como abreviatura de New Technology (sigla empleada por ejemplo en el sistema operativo Windows NT). Así, la frase podría traducirse como "Esta es poesía tecnológicamente nueva". En cualquier caso, se trata de un enunciado autorreferencial, clave –como adelanté– en la concepción de la pieza artística de Corredor Parra. Si los cuadros de Magritte ponían en escena un cuestionamiento de la idea misma de representación, la frase que rige el poema se instala, a través de la deixis del pronombre demostrativo, en el orden de la negatividad, en tanto al desvanecerse niega la existencia de un afuera del texto.

Una vez declaradas y asignadas las variables globales, resulta imprescindible el conocimiento de la plataforma Processing, para comprender que se ejecutarán las funciones estándar *setup* y *draw*, en ese orden. La función *setup* asigna la "idea" a la variable *draft* ("borrador"), operación que connota una concepción de escritura (o de pensamiento) como proceso sujeto a correcciones.

```
void setup(){
 draft=idea;
 Write();
 ReThink();
```

```
}
```

Enseguida, la función llama a otras dos, definidas por el programador, (*Write*, "escribir" y *ReThink*, "reformular"), rompiendo con la lectura lineal del *code poem* para imponer otra, como veremos, iterativa.

La función *Write* imprime en la pantalla implícita la variable *draft*, que contiene la idea original completa "This is'nt p0etry". Pero de inmediato, *ReThink* ejecuta un ciclo *for*, cuyo código (sin dudas más oscuro para el lector poco avezado) descompone la cadena *idea* en un vector que almacena una por una sus letras, lo que permite al algoritmo en lo sucesivo trabajar con la frase grafema por grafema.

Una vez completadas estas operaciones se ejecutará entonces, de modo cíclico, la función *draw*, hasta que un *noloop* la interrumpa y finalice el desciframiento del poema de código. La función *draw* solo efectúa una tarea, el llamado a otra llamada "ReWrite" ("ReEscribir"). En ella se encuentra el núcleo conceptual de la pieza de Corredor: las letras de la frase original se irán reemplazando de modo gradual y aleatorio (gracias a la función *random*), por espacios en blanco, para formar ocasionalmente nuevos términos o desestructurar sintáctica y léxicamente la oración, hasta que solo quede un punto final. La lectura del código, sin embargo, permite advertir que este procedimiento se encuentra enunciado de forma metapoética, ya que la variable *poetry* guarda el verso progresivamente escindido:

```
if(poetry.equals(silence)){
 println("."); noLoop();}
```

Cuando poesía es igual a silencio, entonces *imprimir* un punto final e *interrumpir* la ejecución del ciclo.

Existe un "Ars poetica 2.0" que puede visualizarse y ejecutarse en el perfil de Alejandro Corredor Parra[13] de la plataforma Processing que, entre otras características, permite "bifurcar" (*fork*) un proyecto para apropiárselo y modificarlo con libertad. Se trata de una reescritura del código original publicado en 2012 que, entre otras alteraciones, reformula la cadena idea como "This no es p0etría.", empleando una suerte de spanglish:

```
var silence = "                    ";
var idea = "This no es p0etría.";
var draft;
var poem = [];
var ego;
var mood = 80;
var myPixelInk = 'Courier New';

function setup() {
 ego = windowHeight;
 createCanvas(ego, ego);
 page();
 draft = idea;
 Write();
 ReThink();
}

function draw() {
 background (mood);
 ReWrite();
}
```

[13] https://openprocessing.org/sketch/564524

```
function Write() {
 text(draft, ego/2, ego/2);
 print (draft);
}

function ReThink() {
 poem=split(draft, '');
}

function ReWrite() {
 var seek = byte(random(0, poem.length));
 poem[seek] = " ";
 var poetry = join(poem, '');
 text(poetry, ego/2, ego/2);
 print(poetry);
 if (poetry == silence) {
  text(".", ego/2, ego/2);
  print(".");
  noLoop();
 }
}

function page() {
 background (mood);
 frameRate(ego/ego);
 textAlign(CENTER, CENTER);
 textFont(myPixelInk);
 textSize(mood/2);
}
```

Esta nueva versión del código incorpora las variables *mood* y *ego*, ausentes en la versión impresa, cuya connotación semántica sugiere la presencia de una suerte de subjetividad maquínica. En el programa, la variable *mood* ("ánimo") controla el tamaño de la tipografía *Courier New* y el color de fondo de la página sobre la que se imprime la "idea". A su vez, *ego* se asigna sugerentemente con el alto en pixeles de la ventana del navegador y se emplea en varias partes del código para determinar coordenadas y la frecuencia de ejecución de la función *draw*. Las referencias a la página (*page*) y la tinta (*ink*) complementan el campo semántico asociado a la escritura, más allá del soporte empleado. La ejecución de "Ars poetica 2.0" permite visualizar en la interface web la disolución aleatoria y en tiempo real de la consigna formulada (figura 2), evento que el programador reconstruye mediante operaciones mentales a partir de la decodificación de la versión impresa:

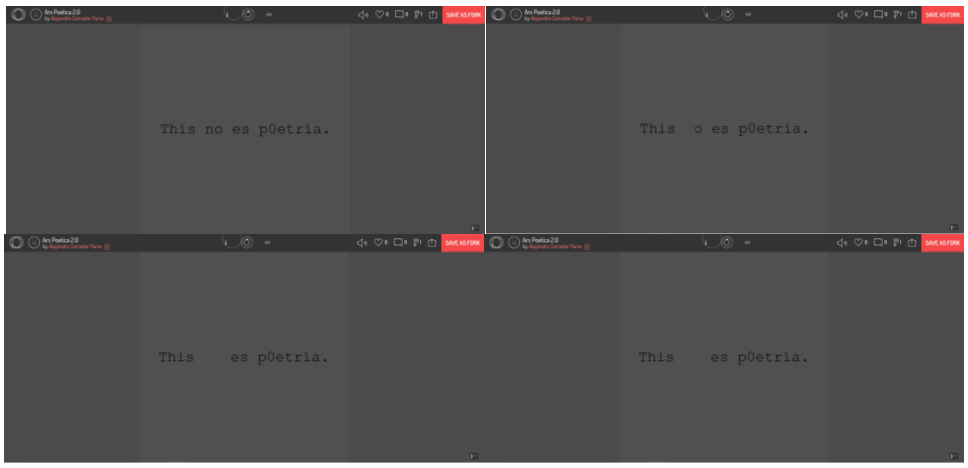

Figura 2

Mientras Magritte ponía en abismo sus pipas jugando con el efecto paradojal de la leyenda "Ceci n'est pas une pipe", la pieza de Corredor Parra –sentadas las diferencias del caso– genera un efecto similar a partir del contraste entre el código (nacido como poema en papel) y la proposición "This no es p0etría", que puede discernirse en relación con aquel o bien autorreferencialmente, en cuyo caso su desmembramiento gradual añade otra capa de significaciones.

Si bien resulta indudable que la lectura de un texto como "Ars poetica" requiere un lector activo e incluso especializado capaz de decodificarlo –en términos ya no metafóricos o comunicativos, sino literales– ¿acaso no ocurre lo mismo con el discurso poético? En este sentido, en el prefacio a *code {poems}*, Jamie Allen sostiene que "Poetry and computer code come out of language. Many forms of poetry can be thought of as code; sets of encrypted verses, to be read and reread, interpreted, «compiled» in the mind" [La poesía y el código informático están hechos de lenguaje. Muchas formas de poesía pueden considerarse como un código; conjuntos de versos encriptados, para ser leídos y releídos, interpretados, "compilados" en la mente] (citado en Bertran, V).

Consideremos entonces, por citar un caso emblemático, los dos versos iniciales de la estrofa XIII de la *Fábula de Polifemo y Galatea* (1627) de Luis de Góngora: "Ninfa, de Doris hija, la más bella / adora, que vio el reino de la espuma". Si admitimos que la lectura de *code poetry* demanda saberes previos específicos, no es menos verdadero que textos como el de Góngora requieren para su exégesis (al menos) conocimientos de mitología y tropos, imprescindibles no solo para reconocer las fuentes mitológicas del poema, sino recursos como el hipérbaton ("Adora [a una] ninfa hija de Doris, la más bella que vio el reino de la espuma") o la sinécdoque presente en "el reino de la espuma", que remite al mar.

Como advertimos a partir de la lectura de "Ars poetica", los lenguajes de programación, en líneas generales, demandan una sintaxis estricta y carente de ambigüedades, y no suelen incluir como parte de su léxico inherente fijo sintagmas adjetivales, ni preposicionales, ni determinativos. En cambio, incorporan verbos, sustantivos y ocasionalmente adverbios. A la vez, la posibilidad de introducir cadenas entrecomilladas y de nombrar adánicamente variables, constantes y funciones (entre otros elementos), sumado a su potencial performativo en la máquina, promueve la emergencia de prácticas de escritura literaria experimentales, que exigen profundizar la alfabetización digital y la creación de marcos teóricos apropiados, para los que los ECC constituyen un aporte ineludible.

}

Obras citadas

Bertran, Ishac. *code {poems}*. Ishac Bertran, 2012.

Cox, Geoff y McLean, Alex. *Speaking Code: Coding as Aesthetic and Political Expression*. The MIT Press, 2012.

Dällenbach, Lucien. *El relato especular*. Visor, 1991.

Gache, Belén. *Escrituras nómades. Del libro perdido al hipertexto.* Ediciones Trea, 2006.

Marino, Mark. "Critical Code Studies". *Electronic Book Review,* 2006. https://electronicbookreview.com/essay/critical-code-studies/

Marino, Mark. *Critical Code Studies*. The Mit Press, 2020.

Raley, Rita. "Code.surface| |Code.depth". *Dichtung Digital. Journal für Kunst und Kultur digitaler Medien*. Nr. 36, Jg. 8, 2016.