ASRI ARTE Y SOCIEDAD REVISTA INVESTIGACIÓN
Número 15 Julio 2018
Número coordinado por Interfaces Culturales - Arte y Nuevos Medios [Grupo Investigador UCLM]

EL ARTE DE LAS COSAS

THE ART OF THINGS

Lino García Morales
Victoria Gutiérrez Colino
Universidad Politécnica de Madrid (España)

Recibido: 10 de mayo de 2018
Aceptado: 7 de julio de 2018

Resumen:

El arte de los nuevos medios está íntimamente ligado al código. El código no es sólo un recurso teleológico sino es, en si mismo, un recurso estético. El hombre ha superado la «era de la información» y ha alcanzado la «era del conocimiento». Ahora dispone de datos (grandes y pequeños) para aprender de ellos: información consumida, producida, almacenada y transmitida por cosas "inteligentes". El presente y el futuro se mide en bytes. El futuro es digital y el arte de las cosas es su principal soporte estético.

Palabras clave: *Internet de las Cosas, Arte de los Nuevos Medios, Código.*

Abstract:

New media art is inextricably linked to the code. The code is not only a teleological resource but the in itself an esthetical resource. The man has overcome the «information age» and has reached the «knowledge age». You now have big and small data to learn from them: information consumed, produced, storage and transmitted by "smart" things. The present and the future are measured in bytes. The future is digital and the art of things is a main esthetical support.

Keywords: *Internet of Things, New Media Art, Code.*

* * * * *

1. Cosas

Una *cosa*, en este ámbito, es un *sistema* capaz de contener, procesar y transmitir información. Cualquier *cosa* está compuesta de una parte dura (*hardware*), de una parte blanda (*software*) y otra parte híbrida (*netware*). Las *cosas* son números: códigos, signos, que representan a otras cosas; números que, según John von Neumann,

significan cosas y *hacen* cosas. Ésta distinción, precisamente, determinó el nacimiento del *software*. Estas *cosas* tienen el poder de modificarse a sí mismas[1].

Los ingenieros que diseñaron estas primeras *cosas* lo hicieron a partir de un presupuesto brillante (por necesidad), al que cada vez la industria le ha restado más importancia, y es el de conseguir sistemas fiables a partir de partes no fiables. El consejo de John von Newmann era claro y conciso: No inventar nada. Esta arquitectura de la necesidad autoimpuesta hizo posible el origen de las máquinas *hardware* y *software* y, posteriormente, del *netware*: la combinación *hardware-software* destinada al transporte de la información.

Los *datos* e *instrucciones* se mezclan en la máquina, son números (*software*) que el *hardware* convierte en señales eléctricas y viceversa. En un principio el *hardware* se "programaba" configurando los circuitos mediante cables externos de la misma manera que las máquinas mecánicas se reconfiguran a través de algún mecanismo. Turing fue uno de los pioneros en observar que: "era posible inventar una sola máquina que pueda utilizarse para computar cualquier secuencia computable"[2]. Según Julian Bigelow, el ingeniero jefe del Proyecto del Computador Electrónico, la importancia de esta observación de Turing es que la estructura resulta trivial. La estructura puede ser sustituida por código. Esa única máquina universal capaz de transformar la información a partir de la propia información alberga el concepto más extendido de *hardware* hoy día, mientras que esas máquinas en forma de secuencias computables son entendidas como *software*. Turing demostró que era posible codificar una máquina como un número y decodificar un número como una máquina. Von Neumann determinó cuál sería el conjunto mínimo de esos números que, desempeñando cada uno una función simple, permitieran crear programas complejos sin necesidad de modificar la máquina física. Éste mínimo conjunto de instrucciones permitió el ensamblaje de máquinas de Turing en sistemas que no son máquinas de Turing.

Una cosa transforma unos datos en otros a través de instrucciones. Esta transformación se denomina computación. Las cosas *computan* y *copian*. La comunicación, el *netware*, en realidad no mueve físicamente nada, sino que produce una nueva copia en otra parte. Una máquina de Turing puede hacer copias de sí misma.

La máquina de Turing fue la semilla de la virtualización. Un ordenador puede hospedar múltiples máquinas virtuales. Se podría decir que cada aplicación es una máquina virtual, aunque el concepto de máquina virtual se entiende más como el hospedaje de un sistema operativo en otro. Las aplicaciones en la nube están direccionadas por una IP que funciona similarmente a como lo hace el contador de programa en la memoria local. Son una ampliación distribuida de la propuesta inicial de von Neumann. De la filosofía «HAZ ESTO CON ESO» y, gracias al *netware*, la computación evoluciona a una arquitectura redundante de partes no fiables, tal y como lo pensó von Neumann, de «HAZ ESTO CON ALGO COMO ESO». El futuro del código digital, según Dyson, requiere comportarse como el ambiguo código genético: transcripción exacta, pero expresión redundante. No se trata de modelos de procesos inteligentes sino de procesos inteligentes.

[1] DYSON, George. *La catedral de Turing. Los orígenes del universo digital*. DEBATE, Barcelona, mayo 2015, p. 108.
[2] DYSON. *La catedral de Turing*, p. 108.

2. Arte de las Cosas

Los pioneros del arte del software (*software art*), programación artística o arte del código no sólo estaban interesados en explorar las posibilidades estéticas y generativas del *software* sino en la propia estética del código; algo a lo que tímidamente se llamó «arte algorítmico» en sus inicios a principios de 1965: el arte de programar flujos de trabajos, procesos o reglas con un ordenador. Según Pau Alsina:

> Hoy el *software art* se basa en la consideración de que el *software* no es tan sólo un instrumento funcional, sino que también se puede considerar una creación artística en sí misma: el material estético resultante es el código generado y la forma expresiva es la programación de *software*. Concretamente podemos distinguir dos líneas de trabajo básicas: de una parte, el trabajo con el código, es decir, en este caso el material de base son las instrucciones formales o código y, de otra parte, el trabajo crítico en torno a la vertiente cultural del *software*, o sea, la intervención sobre la concepción del *software* entendido como artefacto cultural, que actúa dentro de la cultura […][3]

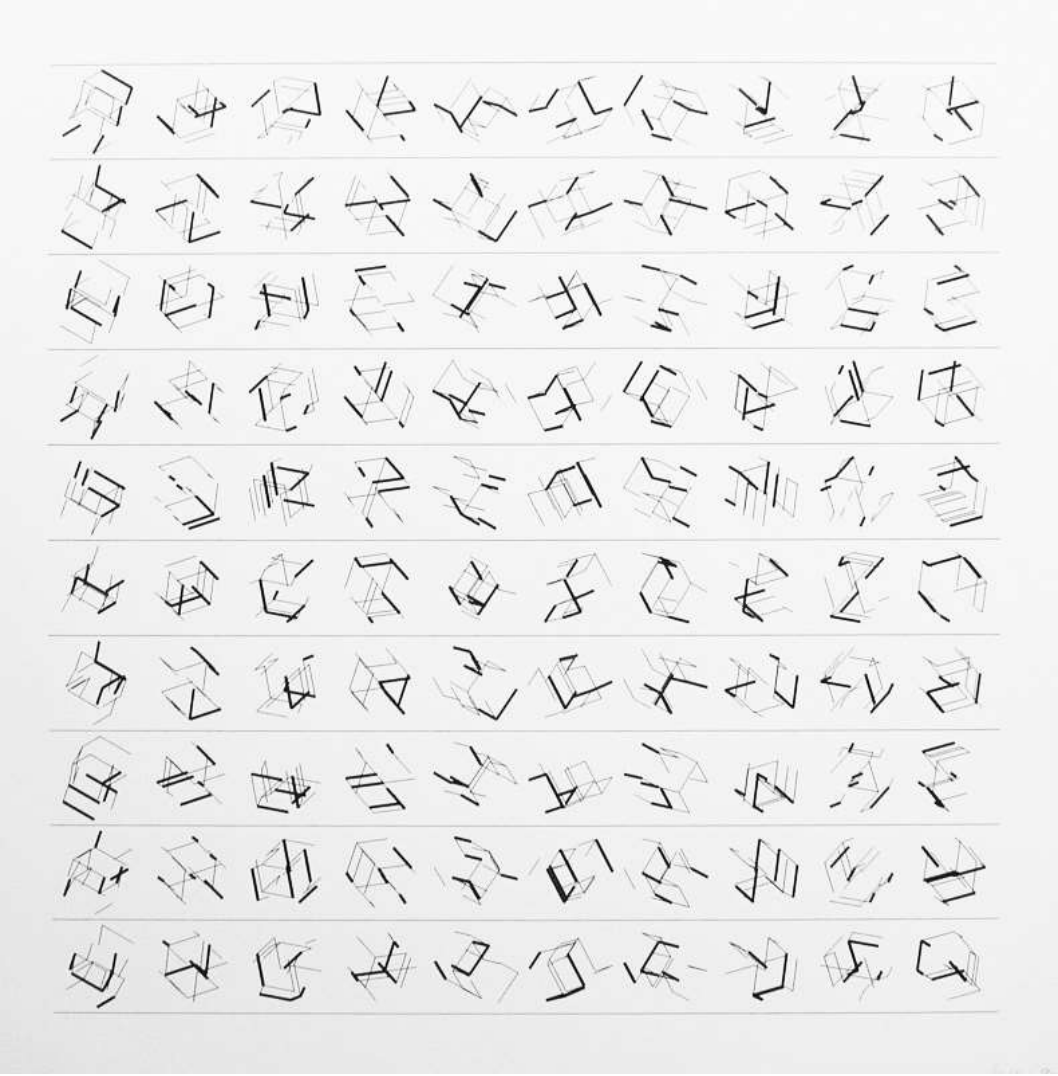


Figura 1. Manfred Mohr, *P455a*, 1990.

[3] ALSINA, Pau. *Arte, ciencia y tecnología*. UOC, 2007, pp. 67–68.

Al menos desde la línea de trabajo con el código, el propio código puede ser juzgado estéticamente o quizá filosóficamente, acorde a la liquidez de los tiempos que corren. El código es lenguaje, de hecho, un lenguaje formal. Para que el ordenador haga algo es necesario escribir un código. El código no es más que un conjunto de símbolos y reglas sintácticas y semánticas que definen la estructura y el significado de todos sus elementos y expresiones, y es utilizado para controlar el comportamiento físico y lógico de una cosa.

Escribir un código, como expresa el poeta y experto en inteligencia artificial Francisco Serradilla, en el prólogo de su *Tratado inusual del universo*, es "de letras".

> Escribir un programa de ordenador es como escribir un poema: hay algo que se quiere expresar –a veces no se sabe bien qué–, un lenguaje para intentarlo e infinitas maneras de hacerlo, unas más claras que otras, unas más estéticas que otras. Así que la computación es un refugio más para el impulso creador, y ahí es donde llevo algunos años refugiado componiendo "sonetos prácticos"[4].

En su ensayo *Software: ¿arte?*, el creador de Wiring, antecesor directo de Arduino, Hernando Barragán, ofrece una visión similar: "*software art* significa un cambio en el artista desde su visión de pantallas a la creación de sistemas y procesos en sí mismos". Esos sistemas y procesos en sí mismos son resultado de esos "sonetos prácticos".

> En el mundo de los bits, o mundo digital ocurren muchos procesos en tamaños y velocidades cuyas escalas están bastante lejos de nuestro sentido de la experiencia. En el transcurso de nuestra interacción con las representaciones digitales, la mayoría de nosotros no tenemos percepción de los bits, estructuras de código, algoritmos, o siquiera de recordar que existen. Es aquí donde podemos ver que la experiencia de la representación digital necesita ser aún más explorada. Pero estas operaciones, aunque son imaginables, son en sí mismas inalcanzables a nuestros sentidos, y más aún en éstas hay muy poca semejanza cuando las comparamos con la forma como nosotros interactuamos con el mundo físico, sólo al final somos capaces de percibir sus efectos. En el mismo instante en que estas operaciones trabajan de manera invisible detrás de escena, realizando estas transformaciones, parece un acto de magia, y es así como el medio a través del cual se posibilita el cambio no tiene forma en sí mismo y, es entonces, operación pura. Provee una función sin pérdida ni ganancia en el proceso. En este sentido los medios digitales se perciben inmateriales, parecen pura función, intocables por la forma. El código es entonces un lenguaje con posibilidades estéticas y de forma, una codificación que tiene un poder transformador que en el medio digital opera transformaciones, que a su vez podrían generar nuevas formas[5].

El código es una forma de expresión de la cultura contemporánea; de hecho, para Barragán, el código es la forma más pura del pensamiento creativo contemporáneo. Código es arte. Si admitimos esto, debemos aceptar una estética del código y la posibilidad de lo «bello» y lo «feo», del mal gusto y del buen gusto, etc. De la misma manera debemos aceptar una ética del código y el reconocimiento de lo «bueno» y lo «malo».

Sin embargo, cuando se trata la estética del código, se suele confundir ética y estética. Un buen código es el que funciona. Se escribe para resolver un problema y debe

[4] SERRADILLA, Francisco. *Tratado inusual del universo*. Colección Poesía en Madrid, Madrid, 2000, p. 8.
[5] BARRAGÁN, Hernando. "Software: ¿arte?". Disponible online en: http://www.banrepcultural.org/artenlared/sccs/02/barragan.pdf (Fecha de consulta: 10/05/2018)

hacerlo; incluso puede que de manera eficaz. Sin embargo, no tiene por qué ser bello. Por otra parte, no puede ser bello un código que no funciona; que carece de utilidad. No puede ser bello porque la belleza radica en sus posibilidades semánticas, no sintácticas. En la doble naturaleza del código está la contradicción. El arte puede ser inútil; el código, en su esencia técnica, no. Su última finalidad es representar; sin embargo, la finalidad del código es procesar, transformar la información de la misma manera que la finalidad de una ecuación matemática, es explicar o resolver un problema. La belleza está condicionada, de alguna manera, por la economía con que satisface su finalidad, por la capacidad de abstracción, por la simplicidad de comprensión, etc.

En general es deseable que un código sea:

Inteligible o leíble por cualquiera que no haya escrito el código[6].
Mantenible o fácil de evolucionar.
Simple, sin complicaciones innecesarias, ni redundancias[7].
Eficiente, sin consumir recursos innecesariamente: memoria, capacidad de proceso, tiempo de ejecución, etc.
Claro o fácil de entender sin comentarios: auto–documentado, nombres adecuados de métodos y atributos, fragmentado en códigos pequeños, lógicos, estructurados, escalables, modulares, independientes, etc.

Se suele pensar que un buen código es, por lo tanto, *legible*, *mantenible*, *simple*, *eficiente* y *claro*. De igual modo se suelen atribuir los mismos atributos a un código bello, realmente hermoso.

Los programadores llaman «código espagueti» a aquel donde todo vale con tal de resolver el problema. Es probablemente el tipo de código que satisface el menor número atributos deseables. En este tipo de código la ausencia de estructuras profundas u ordenamiento planificado es evidente. La economía no importa; sólo la capacidad de satisfacer determinada transformación. Suele ser un código ininteligible e inmantenible, no tiene por qué ser claro, puede llegar a ser complejo, etc.

En el otro extremo, es posible un código exquisito, en el que todo detalle es tenido en cuenta. Suponen el canto del cisne en cuanto a las posibilidades que brinda un sistema. La belleza llega aquí, según el programador Carlos Benítez, desde la elegancia y el adorno, independientemente de la claridad de su exposición final. Se suele asociar al academicismo.

Entre ambos extremos de pretensión es posible encontrar la mayoría del código que se produce a diario (quizá con mayor tendencia a la baja producida por la atracción del prototipado rápido). El propio Benítez intenta ofrecer una explicación sistémica a esta

[6] Según Martin Fowler: "Cualquier tonto puede escribir código que un ordenador pueda entender. Los buenos programadores escriben código que pueden entender los humanos." Donald Knuth, matemático y profesor emérito en la Universidad de Stanford, va aún más lejos. No sólo debe ser legible sino además su lectura debe producir placer.

[7] Esto es quizá, lo más importante para Robert C. Martin: el pragmatismo final frente, por ejemplo, la habilidad estructural. Estadísticamente un desarrollador debe pasar menos tiempo escribiendo código que leyéndolo y no al revés. Es necesario optimizar la lectura, aunque ello requiera un mayor esfuerzo de redacción. Para Martin lo esencial es que el código sea "limpio". Para ampliar, ver: MARTIN, Robert C. *Clean Code. A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

escala de belleza en función de las dimensiones[8] lo que supone reducir lo bello a lo complejo.

Un código bello es, esencialmente, un código bueno y económico. Es cierto que la belleza se muestra en todo su esplendor en la complejidad; en la medida en que se consigue lo complejo desde lo simple. La belleza está en lo esencial independientemente de la escasez o la abundancia. Existen infinitas formas de viajar desde un punto A a un punto B pero sólo una de esas formas es la más corta independiente de lo lejos que esté A de B.

Linus Torvalds, creador del sistema operativo Linux, explica a través de un pequeño ejemplo la diferencia entre un código programado con "mal gusto" (Código 1) de uno programado con "buen gusto"[9] (Código 2). Para evitar una explicación escabrosamente técnica, la diferencia entre ambos códigos estaba en el tratamiento de las condiciones fronteras o situaciones especiales (que aparecen codificadas mediante sentencias `IF ELSE`).

```
// Inicializacion de los bordes de una reticula
final int GRIDSIZE = 16;
int[][] grid = new int[GRIDSIZE][GRIDSIZE];

for (int r = 0; r < GRIDSIZE; ++r) {
   for (int c = 0; c < GRIDSIZE; ++c) {
      // Top Edge
      if (r == 0)
         grid[r][c] = 0;
      // Left Edge
      if (c == 0)
         grid[r][c] = 0;
      // Right Edge
      if (c == GRIDSIZE - 1)
         grid[r][c] = 0;
      // Bottom Edge
      if (r == GRIDSIZE - 1)
         grid[r][c] = 0;
   }
}
```

Código 1. Ejemplo de "mal gusto".

El código de "mal gusto" (1) trata las condiciones fronteras explícitamente mientras el código de "buen gusto" (2) lo hace implícitamente. Dicho de otra manera el código de buen gusto no distingue torpemente entre una situación específica y una general y esto reduce un código de 10 líneas a otro tan sólo de 4[10] mucho más simple y eficaz. En esencia el código de Torvalds elimina cualquier redundancia, se queda

[8] BENÍTEZ, Carlos. "La estética del código fuente: a la búsqueda del arte en la programación". Disponible online en: http://www.etnassoft.com/2016/09/16/la-estetica-del-codigo-fuente-a-la-busqueda-del-arte-en-la-programacion/ (Fecha de consulta: 10/05/2018)

[9]BARTO, Brian. "Applying the Linus Torvalds 'good taste' coding requirement". Disponible online en: https://medium.com/@bartobri/applying-the-linus-tarvolds-good-taste-coding-requirement-99749f37684a#.xct244qiq (Fecha de consulta: 10/05/2018)

[10] El número de líneas no incluye la línea de definición de la retícula, necesaria en ambos códigos; ni la definición de una constante para parametrizar las dimensiones del problema.

en lo esencial y además, se entiende, es claro, es eficiente; es el camino más corto, intelectualmente más relevante, entre A y B. El primero lo puede hacer cualquiera, el segundo no, sólo un buen programador, un artista, produce un código de esta naturaleza; de esos que provoca una inquietante reflexión: ¿por qué no se me había ocurrido con lo simple que es?


```
// Inicializacion de los bordes de una reticula
final int GRIDSIZE = 16;
int[][] grid = new int[GRIDSIZE][GRIDSIZE];

for (int i = 0; i < GRIDSIZE; ++i) {
   // Top Edge
   grid[0][i] = 0;
   // Bottom Edge
   grid[GRIDSIZE - 1][i] = 0;
   // Left Edge
   grid[i][0] = 0;
   // Right Edge
   grid[i][GRIDSIZE - 1] = 0;
}
```

Código 2. Ejemplo de "buen gusto".

El código 1 es un ejemplo de un código de “mal gusto”. La función de este código es inicializar los bordes de un arreglo bidimensional a 0. Observe que este código resulta complejo: tiene dos lazos o bucles (en la mayoría de los lenguajes de programación los bucles anidados son mucho más lentos e ineficientes) y para detectar los bordes emplea cuatro condicionales. Si el valor de `GRIDSIZE` fuese, por ejemplo, 64 serían necesarias 4096 iteraciones para inicializar sólo los 256 puntos del borde. No es eficiente, no es simple; funciona, pero no es bello; al menos no está escrito con “buen gusto”.

El código 2 hace lo mismo que el código 1 pero con “buen gusto”. Como puede observar tiene un solo bucle que, además, sólo realiza 64 iteraciones sin ninguna condicional. Ambos códigos son “buenos”, no son erróneos, funcionan, pero sólo uno de los dos expresa la misma acción con tanta simplicidad, mantenibilidad, inteligibilidad, claridad y eficiencia. Sin embargo, lo que hace es bien simple, profano, rutinario, accesorio.

El desafío principal de la ingeniería software está en acortar los tiempos de desarrollo, mejorando la mantenibilidad y calidad del código. Un «buen» código debe ser simple, mantenible, inteligible, claro, eficaz. Todas estas propiedades están relacionadas pero un código no mantenible, incluso cuando funciona bien, tiene poco o ningún valor; está condenado a desaparecer. La norma IEEE 19990 define la mantenibilidad como “la facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno”. El mantenimiento correctivo es análogo a la restauración mientras que el mantenimiento predictivo es análogo a la conservación.

Observe ahora la siguiente línea de código.

```
10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

Este código, escrito en lenguaje BASIC para la máquina Commodore 64, es, a la vez, el título de un libro de varios autores publicado por MIT Press que analiza el fenómeno de la computación creativa[11] y es capaz de generar un valor aleatorio entre 0 y 1 con la función `RND()` al que suma `205,5`. El resultado es una cadena alfanumérica (`CHR$`) redondeada automáticamente, de modo que puede ser `205` ó `206`; lo que corresponde a los caracteres / o \. La repetición, mediante la operación (`GOTO 10`) imprime uno tras otro los caracteres en pantalla, generando un patrón aleatorio de líneas diagonales que forma una especie de laberinto cada vez diferente.

Figura 2. Captura de 10 PRINT CHR$(205.5+RND(1)); : GOTO 10 en acción.

La magia de esta belleza exige un conocimiento al espectador, no sólo al artista, para poder apreciarla. El código no puede brillar, sin más. Desde el punto de visa holístico que propone Hofstadter, el poeta algorista[12] parte de un conjunto finito de bloques, un repertorio, y un conjunto de reglas, ligaduras, con los que construir superbloques, macrobloques conceptuales, abstractos, que encajen entre sí a diferentes escalas hasta alcanzar el *etéreoware* (los conceptos puros que están detrás del *software*)[13]. La creación tiene muchos grados de libertad y el «orden» importa. Las ligaduras restringen las alternativas, reducen el espacio "solución" disminuyendo los grados de libertad. En su texto *Computación creativa y otros sueños* Serradilla lo define mejor:

> En una ecuación, los grados de libertad son el número de variables independientes, es decir, las magnitudes que pueden variar en el espacio del problema. En robótica, estos grados de libertad se corresponden con la posición de cada uno de los ejes del robot, y determinan la ubicación del extremo final (la pinza) del brazo robótico.

[11] MONTFORT, Nick *et al.*: *10 PRINT CHR$(205.5+RND(1)); : GOTO 10*. MIT Press, 2012.
[12] Algorista es la persona que cultiva la ciencia de los algoritmos.
[13] HOFSTADTER, Douglas R. *Gödel, Escher, Bach. Un eterno y grácil bucle*. Tusquets, 4ta Edición, 2014, p. 426.

> A mayor número de grados de libertad, mayor variedad en las situaciones posibles, mayor complejidad en el sistema, aunque no necesariamente mayor diversidad en los resultados, ya que la variable independiente (la posición de la pinza) quizá tome el mismo conjunto de valores válidos a partir de cierto número de grados de libertad.
> La Ingeniería consiste en última instancia en la restricción de alternativas mediante el establecimiento de reglas, protocolos y normas de desarrollo de soluciones a los problemas, para dejar pocos grados de libertad: el ancho y el largo de un puente, su altura y el peso máximo soportado, por ejemplo, lo que nos permite calcular sus requisitos estructurales, la cantidad de material a utilizar y el tiempo y personal necesario para su construcción.
> Curiosamente, a los humanos les es más fácil crear si el conjunto de posibles "libertades" se estrecha. Por eso surgen estilos, corrientes, academias, modas. Pareciera, además, que cuando la libertad crece por algún lado –véase en el flamenco libertad de las fluctuaciones melódicas del cantaor, o en el jazz la libertad en la improvisación– se restringe por otro –la rigidez rítmica y armónica en cada palo flamenco, las cadencias fijas y estrictas en el jazz–, como para mantener constante el grado de complejidad de la obra.
> En Literatura se escriben normas, se categorizan estilos: se analiza la estructura del cuento o de la novela, se pontifica sobre el objetivo de la poesía y sobre lo que está o no permitido o recomendado. En muchos "cursillos rápidos" de literatura se "enseña" a escribir. Así nos va: se han creado hordas de escritores que siguen el mismo patrón con más o menos oficio y aburren hasta a sus profesores.
> Los grandes creadores, quizá, son aquellos capaces de manejar su medio con mayor número de grados de libertad; los que no necesitan cánones, normas ni estilos, los que se mueven en un espacio del problema con infinitas dimensiones, y en estos espacios terribles encuentran una de las soluciones de la ecuación artística[14].

El arte de las cosas se produce y reproduce en el mundo del arte casi siempre oculto, provocando sospecha a la par de fascinación. Los nuevos medios son las nuevas pizarras donde leer no sólo los "sonetos prácticos" sino sus evocaciones. No todos los lenguajes son igual de bellos en la misma medida en que no todos los lenguajes son igual de simples, claros, inteligibles e incluso eficientes; pero, con mejor o peor "gusto", con más o menos líneas, con mayor o menor esfuerzo, el algorista debería poder conseguirlo o al menos imitarlo[15].

A pesar del «arte de las cosas» la abrumadora mayoría de las obras de arte de los nuevos medios donde el código juega un papel relevante, no se exponen como código en sí. Lo que percibimos no es la causa sino el efecto. Está claro que existe una relación causa–efecto pero, como hemos mencionado antes, existen múltiples formas de viajar desde A hacia B. Artistas como Lozano-Hemmer empiezan a cambiar la relación, hasta ahora conservadora, del artista con el código. En *Pseudomatismos*, primera exposición monográfica de su obra en el MUAC, además del catálogo, se publicó en una memoria USB todos los códigos fuentes utilizados para programar las piezas de la exposición[16]. Cualquier programador, artista, o Restaurador tiene acceso a los algoritmos y métodos que los ingenieros de su estudio desarrollaron para cada obra, en lenguajes C++,

[14] SERRADILLA, Francisco. Creatividad, arte, ingeniería y grados de libertad. Libro de Notas, diciembre 2006. Disponible online en: http://librodenotas.com/computacion/10225/creatividad-arte-ingenieria-y-grados-de-libertad (Fecha de consulta: 10/05/2018)

[15] La traducción de un algoritmo de un lenguaje a otro tiene un símil en la traducción de un idioma a otro: el contenido es el mismo pero el *kit* de elementos y reglas es diferente. Los lenguajes de computación no son tan ricos en repertorio como los lenguajes naturales (de hecho, son más bien redundantes); lo que supone una ligadura auxiliar.

[16] Se podría decir que este gesto inaugura la posibilidad de disfrutar, o no, de la estética del código.

OpenFrameworks, Processing, Delphi, Assembler y Java. Es la primera exposición de arte de código abierto.

3. La sustancia de las Cosas

El siguiente código, escrito en Processing por Jim Plaxco, produce la misma representación simbólica de la obra *Schotter*, de Georg Nees. Nees es probablemente el primer artista que presenta dibujos generados algorítmicamente por un ordenador digital, bajo control de un programa. Este es un ejemplo paradigmático de la estética generativa basada en el principio estocástico y en la redundancia estética.

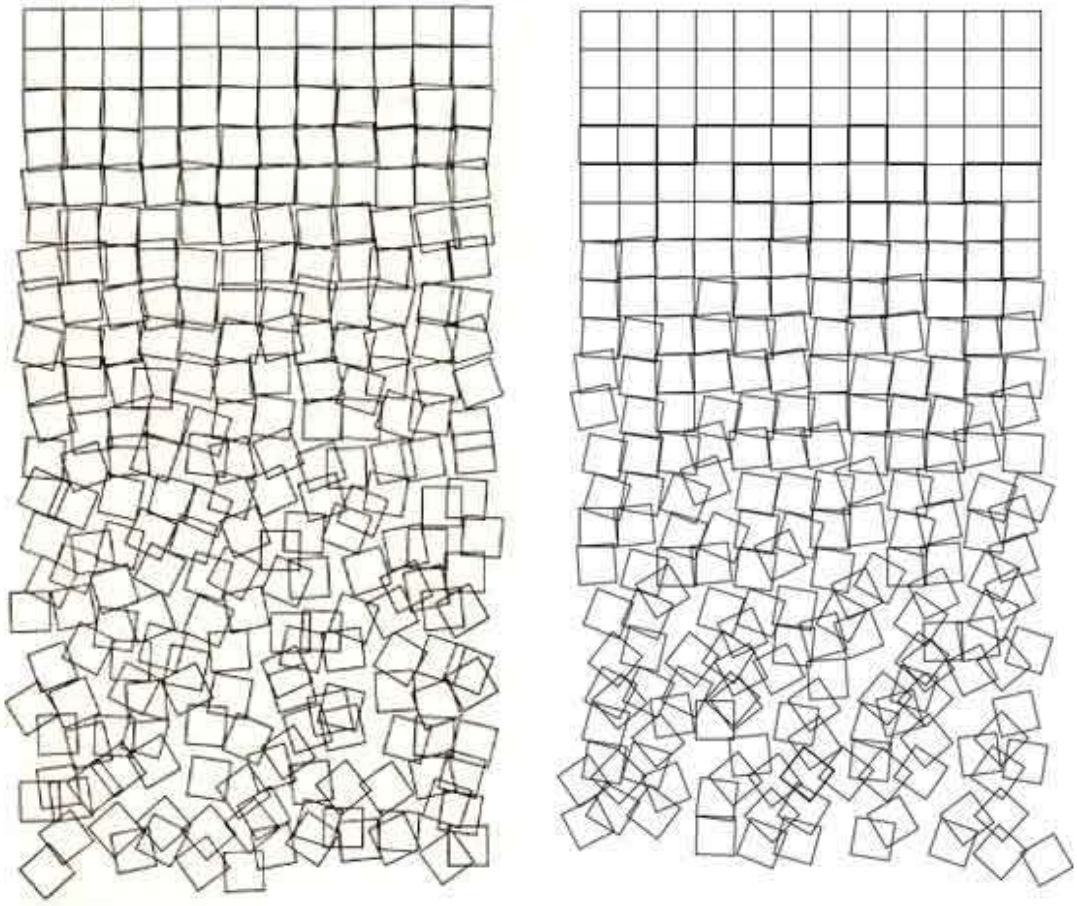

```
// Georg Nees, Schotter, Reproduction by Jim Plaxco, www.artsnova.com
int columns = 12;      // number of columns of squares
int rows = 22;              // number of rows of squares
int sqrsize = 30;      // size of each square
float rndStep = .22;        // Rotation Increment in degrees
float randsum = 0;     // Cumulative rotation value
int padding = 2* sqrsize;  // margin area
float randval;              // random value for translation and rotation
float dampen = 0.45;        // soften random effect for translation

void setup () {
   size ((columns+ 4)* sqrsize, (rows+ 4)* sqrsize );
   background (255);  // set background color to white
   stroke (0);        // set pen color to black
   smooth ();         // use line smoothing
   noFill ();         // do not fill the squares with
   rectMode (CENTER); // use x,y value as the
   noLoop ();         // execute draw () just one time
} // end of setup ()

void draw () {
   for (int y=1; y <= rows; y++) {
      randsum += (y* rndStep); // Increment the random value
      for (int x=1; x <= columns; x++) {
         pushMatrix ();
         randval = random (-randsum, randsum);
         translate (padding+ (x* sqrsize)- (.5* sqrsize)+ (randval* dampen),
                    padding+ (y* sqrsize)- (.5* sqrsize)+ (randval* dampen));
         rotate (radians(randval));
         rect (0, 0, sqrsize, sqrsize);
         popMatrix ();
      } // end of x loop
   } // end of y loop
} // end of draw ()
```

Código 3. Código en Processing que recrea *Schotter*.

Las Figuras 3 y 4 permiten comparar la producción de Nees en 1960 con una recreación en 2018 a partir del código de Plaxco, programado en 2006.



Figuras 3 y 4. A la izquierda: Georg Nees. *Schotter*, 1960. A la derecha: Jim Plaxco. versión de *Schotter*, 2006. Téngase en cuenta que cada realización es diferente.

Cualquier código emplea las mismas piezas que la innovación: copia, trasformación, combinación y error. Los números de Von Neumann no son más que un conjunto de estas piezas. Todos los códigos posibilitan la *repetición*. La *iteración* y la *recursión* son ambos mecanismos de repetición del código. La repetición es copia y en el arte de las cosas proporciona *ritmo* y posibilita construcciones anafóricas. La *transformación* se basa, fundamentalmente, en la *traslación*, la *rotación* y el *escalamiento* y la *distorsión*.

Estos son básicamente los mecanismos que utiliza Plaxco para recrear, con *bloques* muy simples, la obra *Shotter* de Nees. La *repetición* se produce en dos niveles, horizontal y vertical, a cargo de la instrucción de iteración `for` mientras que la transformación de cada cuadrado se produce mediante los bloques de *traslación* y *rotación* `translate` y `rotate`; una descripción literal del proceso que realizan. Pero esa *traslación* y *rotación* no es fija, sino que depende del *error* (generado por el *bloque* `random` en la variable `randval`) y de un *escalamiento* parametrizado en la variable `dampen`.

Esta es la sustancia de las cosas. El arte de los nuevos medios es arte del código, "sonetos prácticos" alográficos[17] que hacen posible la maravilla. Lev Manovich definió estos cinco principios del arte de los nuevos medios:

[17] Nelson Goodman introduce en su libro *Los lenguajes del arte. Una aproximación a la teoría de los símbolos*, los términos autográfico y alográfico para establecer una distinción entre original y copia. Una

Representación numérica. Los nuevos medios son código.
Modularidad. Los nuevos medios se agrupan en nuevos medios más grandes, a escalas mayores, sin perder su identidad.
Automatización. Es posible suprimir, en parte, la intencionalidad humana.
Variabilidad. Los nuevos medios pueden ser copias idénticas o versiones de sí mismos sin perder su identidad.
Transcodificación. Cualquier medio traducido, convertido, transformado o migrado al dominio digital se convierte, *per se*, en un nuevo medio[18].

El arte de las cosas es arte de los nuevos medios que exige al menos dos principios adicionales:

Interconectividad. Las cosas estás interconectadas. Las cosas son distribuidas.
Evolutividad. Capacidad de adaptación o mutación de las *partes* de un sistema (cosa) sin que afecte su comportamiento como un *todo*. Las cosas están condenadas a evolucionar.

El arte de las cosas tiene cierta vocación de comunidad, reciclaje y antifragilidad; exige un esfuerzo transdisciplinar. Como dice la crítica Roberta Bosco, es en esta área donde se da "la batalla estética y conceptual de este siglo".

obra es autográfica si la distinción entre el original y la copia es importante; si ni siquiera la duplicación más exacta puede considerarse genuina; como es el caso de la pintura. Una obra es alográfica si cada copia puede considerarse original, como es el caso de la música. La pintura es un arte de una sola etapa y la música un arte de dos etapas., pp. 110-111.

[18] MANOVICH, Lev. *El lenguaje de los nuevos medios de comunicación*. Barcelona: Ediciones Paidós Ibérica, 2006, pp. 72-82.

Bibliografía

ALSINA, Pau. *Arte, ciencia y tecnología*. UOC, 2007.

BARTO, Brian. Applying the Linus Torvalds "good taste" coding requirement. *Medium*. Disponible online en https://medium.com/@bartobri/applying-the-linus-tarvolds-good-taste-coding-requirement-99749f37684a (Fecha de consulta: 05-07-2018)

BARRAGÁN, Hernando. "Software: ¿arte?", *Monoscop*. Disponible online en https://monoskop.org/File:Barragan_Hernando_2002_Software_arte.pdf (Fecha de consulta: 05-07-2018).

BENÍTEZ, Carlos. *La estética del código fuente: a la búsqueda del arte en la programación. Estnassoft*. Disponible online en: http://www.etnassoft.com/2016/09/16/la-estetica-del-codigo-fuente-a-la-busqueda-del-arte-en-la-programacion/ (Fecha de consulta: 05-07-2018).

DYSON, George. *La catedral de Turing. Los orígenes del universo digital*. DEBATE, Barcelona, 2015.

GOODMAN, Nelson. *Los lenguajes del arte. Una aproximación a la teoría de los símbolos.* Barcelona, Paidós Estética, 2010.

HOFSTADTER, Douglas R. *Gödel, Escher, Bach. Un eterno y grácil bucle*. Tusquets, 2014.

MANOVICH, Lev. *El lenguaje de los nuevos medios de comunicación*. Barcelona, Paidós Ibérica, 2006.

MARTIN, Robert C. *Clean Code. A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

MONTFORT, Nick *et al. 10 PRINT CHR$(205.5+RND(1)); : GOTO 10*. MIT Press, 2012.

SERRADILLA, Francisco. *Tratado inusual del universo*. Colección Poesía en Madrid, Madrid, 2000.

SERRADILLA, Francisco. *Creatividad, arte, ingeniería y grados de libertad*. Libro de Notas, diciembre 2006.

Cómo citar este artículo:
García Morales, L. & Gutiérrez Colino, V. (2018). El arte de las cosas. *ASRI. Arte y Sociedad. Revista de Investigación*, (15), 137-149.