Reutilizando códigos como mecanismo de información y conocimiento: Programación en arquitectura

Reusing codes as a mechanism of information and cognition: Scripting in architecture

Pablo C. Herrera Polo
Universidad Peruana de Ciencias Aplicadas, Perú
pablo@espaciosdigitales.org

ABSTRACT
Differently from other regions in the Planet, since 2010, in Latin America textual programming language (Rhinoscripting) is being replaced by its visual equivalent (Grasshopper). This is a consequence of our preference for an interactive platform, and because our design problems are not as complex, so we aim to control geometrical problems or aspects belonging to an product scale instead of an architectural one. Problems emerging when creating code could be improved by modifying and reusing existing solutions as a starting point, since learning would not be centered in the object but in the process of creating it, using a suitable instrument.

KEYWORDS: Visual Programming Language; Textual Programming Language; Scripting; Grasshopper; Rhinoscripting

Introducción

A lo largo de la historia de la arquitectura, los procesos manuales o digitales que la generan se han podido almacenar filmando o fotografiando; es decir, archivamos paso a paso el procedimiento. Ambos objetos (cámara de video o fotográfica) no son la herramienta, ni el instrumento que el autor ha utilizado en el proceso de hacer el objeto. Las películas o fotografías son lineales y consecutivas y, a un tercero, sólo lo llevaría a copiar de la misma manera la solución (Herrera, 2011). En ese proceso, también aparece un conjunto de variables además de la idea del diseñador: intervienen personas y parámetros geométricos para el objeto, así como otros aspectos propios del contexto. Por lo tanto, filmar o fotografiar no permite una verdadera reutilización del conocimiento, sino sólo un aprendizaje de la historia del mismo.

De la manipulación directa sobre una geometría, resulta un objeto que guarda, en su hermetismo, todo el proceso que conduce a un resultado concreto. Evans (1995, 170) en *Translations from Drawing to Buildings*, sostuvo que el dibujo a mano (y sucede lo mismo si es en computadora) no logra traducir por completo la idea original del autor, porque interpone un medio que no es con el que construye. Como consecuencia, es imposible reutilizar la misma información: alcanzamos a rescatar de ella sólo una depuración, y dejamos de ver las muchas posibilidades que se fueron perdiendo en el camino para llegar a la solución. Lyon (2007, 12) sostuvo que *"la arquitectura no produce un objeto del que pueda aprenderse"*. Otras disciplinas, en cambio, no sufren de esta carencia. En medicina, por ejemplo, la transferencia de un conocimiento se potencia con publicaciones que describen métodos y procesos, casi como instrucciones que sirven a otros como punto de partida o evaluación.

Las oportunidades que tenemos de guardar una o varias soluciones a un problema de diseño son escasas, porque lo que producimos es un objeto y el proceso de dicha producción se queda en el creador, e indirectamente limitamos a un tercero a reutilizar las soluciones a las que llegamos alguna vez. Por el contrario, si el proceso se guarda en una secuencia (en programación escrita o visual) puede ser útil a otros diseñadores o estudiantes, porque se está utilizando la misma plataforma de trabajo que usó el autor original.

En esta investigación, que es la base para un proyecto futuro, se presenta un panorama general de cómo Latinoamérica se ubica como creciente usuario de la programación visual. Con un caso de estudio retrospectivo de este proceso se resume la realidad de nuestra región a través del reemplazo de la aplicación de técnicas de programación en problemas de diseño por otras que solo definen casos de estudio e instrumentalizan al estudiante.

Antecedentes

Loukissas y Sass (2004:2) sostuvieron que, si el diseño arquitectónico es aceptado como un proceso importante, y no sólo como un objeto, entonces los arquitectos deberían hacer uso de artefactos que incluyan el proceso de diseño. Ellos llamaron a esta propuesta *process object,* y tomaron *The Art of Computer Graphics Programming* de Mitchell (1987) como punto de partida. En ambas se propone usar un lenguaje de programación escrita para codificar la teoría del diseño, es decir, escribir una probable solución a un problema en un lenguaje natural (*pseudocódigo*) con el fin de traducirlo al lenguaje de un software (*script*) para explorar alternativas cuando cambiamos las líneas que corresponden a variables que controlan el problema, modificando línea por línea, sin ver un objeto hasta no tener terminado el código.

"Los representantes de Rhino entraron a nuestra oficina hace unas semanas. Nos enviaron scripts, les hicimos comentarios y se los enviamos de vuelta. 24 horas más tarde los habían puesto en línea. Estamos hablando sobre a quién le van a pertenecer los scripts y Rhino dice: vamos a publicarlos en línea y si quieres que te pertenezcan tómalos." (Kedan, 2009,143). Y es que si el proceso es expuesto, cualquiera puede tomarlo como suyo, aunque en un principio, solo terminemos usándolo como ejercicio práctico. Este enseña a controlar un problema complejo, pero la facilidad para modificarlo es limitada. Una vez puesto en línea, la idea se replicará con ella misma como referente, y aparecerán otras versiones tomando ésta como punto de partida.

La implementación, con un lenguaje de programación escrita (*Rhinoscripting*), continúa extendiéndose en EE.UU., Europa y Asia, porque la necesidad fue generada en la profesión y no desde la academia (Leach, 2010), mientras que en Latinoamérica ha sido promovida por tres grupos académicos (Herrera, 2011), en un proceso que no convive en un taller de diseño, ni es requerido por la industria de la construcción. Al contrario, se utiliza el mismo régimen de implementación que el software interactivo y en muy pocos casos es un aprendizaje obligatorio, siendo una instrumentalización por repetición y adaptación de modelos académicos que van evolucionando según el aprendizaje o experiencia del instructor. Son pocos los estudiantes de la región que hoy usan la experiencia de programación escrita que comenzó a implementarse en el 2006, siendo el caso más representativo, la oficina de Guillermo Parada, *gt2p* en Chile, cuya evolución nació de la necesidad de resolver una problemática que fue el motivo de su aprendizaje desde un principio.

Para Nardi (1993, 5), estos usuarios o *End-User Programmers* (EUP), *"no son ni casuales, ni novatos, ni naif, son gente como químicos, arquitectos, bibliotecarios (...) que tienen necesidades computacionales y que quieren hacer un uso serio de las computadoras pero no están interesados en convertirse en programadores profesionales"*. Y al igual como sucedió en la década de 1990 con la producción de código por EUPs para la industria (Harrison 2004, 5), en arquitectura aparecieron dos grupos de usuarios similares: 1) los amateur (individuos que aprenden por sí mismos programación), que crean o modifican el código partiendo de soluciones creadas o empezando con ejercicios de manuales o talleres; y 2) aquellos que producen código como parte de un quehacer profesional. Para ambos grupos, la implementación de técnicas de programación ha sido difícil: para los primeros, porque de una curiosidad pasan a una frustración al no encontrar soluciones precisas a problemas particulares, y en la mayoría pasan de la programación más complicada y menos interactiva o escrita (*Rhinoscripting* o *RS*) a la visual (*Grasshopper* o *GH*); para los segundos, cada proyecto es un reto y no hay referentes o una solución en internet como punto de partida, solo queda crearlos, y por la complejidad del problema, se prefiere la programación escrita sobre la visual. Ambos grupos comparten problemas de implementación similares a otras especialidades ajenas a la arquitectura.

Robins et al. (2003) describieron como a un estudiante promedio de ciencias de la computación le toma diez años desde ser aprendiz hasta alcanzar una experiencia notable en programación escrita. Desde el punto de vista del proceso, estos estudiantes buscan soluciones a diferentes tipos de problemas y hasta que no encuentren una problemática como especialidad, sólo les queda aprender de su proceso y no de los resultados obtenidos. Al mismo tiempo, varios autores que han investigado la implementación de programación en artes y arquitectura (Andersen et al., 2003; Herrera, 2007; Bueno et al., 2008) demostraron que la mayoría de estudiantes tienen poco interés en las matemáticas, y la programación no es su principal motivación, prefiriendo temas abiertos a otros absolutos y cerrados. McCauley et al. (2008, 93) sostienen que es común que los aprendices tengan dificultad para leer o escribir códigos. A ello, se debe agregar que es común en Latinoamérica la crítica del objeto y no del proceso, y esto no ha facilitado construir una cultura que promueva al estudiante regresar a los patrones o variables de un proceso con el fin de enriquecerlo, porque sólo modifican el objeto. La mayoría de escuelas de arquitectura locales, promueven

el desarrollo de un objeto partiendo de un "encargo arquitectónico", en vez de identificar un problema en el que el estudiante aprenda del proceso y que él mismo podría regresar tantas veces como necesite explorar, lo que haría necesario utilizar una computadora como aliada de su proceso, y así explorar las relaciones de los componentes que la conforman. Por consiguiente, si no se puede aprender de un proceso ni identificar un problema, ¿cómo es posible traducir algo que no es promovido ni motivado por el estudiante? Más aún, si un instructor promueve el problema, y no hay motivación, el estudiante terminará aceptando una solución basada en la experiencia del instructor, lo que limita la aparición de soluciones futuras para problemas con otro referente como punto de partida. Crear código, *"no depende sólo de la sintaxis y del vocabulario, implica la cantidad de posibilidades y formas de solucionar un problema existente"* (Reas, 2010).

Mitchell (2010, 8) propuso que *"podemos aceptar el vocabulario de elementos de un sistema CAD (...) pero, también programar y quebrar estructuras establecidas y convenciones de diseño; si creamos, modificamos o recombinamos fragmentos de código"*. Converso (2008, 21) hace la analogía para controlar un problema de diseño con la experiencia de programación para internet, porque en la práctica, *"la modificación y el reuso de código son por repetición, el primer procedimiento que llegó a ser popular en la web: desde el principio, siempre ha sido posible que cada web muestre el código que la describe, como consecuencia, se usa parte de él, se cambia y se vuelve a usar."* Hoy, la programación escrita y visual es instructiva, no se discute su implementación, ya que existen procedimientos establecidos en manuales impresos o digitales que son parte de la academia y del día a día profesional.

Caso de estudio en el Perú

Se presentan cinco casos (*Grupos A, X, B, C* e *Y*) que resumen cronológicamente el paso de la programación escrita a la visual, con el fin de generalizar la problemática de implementación y crear un grupo de antecedentes que a futuro permitan una implementación sostenible.

Grupo A. La primera experiencia se realizó con *RS* en el año 2008 en un curso libre de 4 días (32 horas) con participantes egresados y profesores de edad promedio entre 24 y 35 años. Los resultados fueron guiados por los instructores y construidos por los estudiantes, o lo que llamo resultados que siguen al instructor (*form follows instructor*). Esto se debió a que la complejidad de los problemas propuestos por los estudiantes no tenía un límite ni de diseño ni espacio; se alentó el resultado con el fin de llevar al límite la exploración de posibilidades, una vez que se tenía el *script*. Para todo el *Grupo A*, fue una experiencia importante, porque les hizo entender el potencial de la programación, aunque en la actualidad no todos hagan uso de ella. El 25% de ellos siguieron maestrías fuera del país y esta base les permitió poder ampliar y potenciar sus estudios tanto en diseño como en fabricación. El problema principal, es que se juntaron dos situaciones en un mismo espacio de aprendizaje: se intentó que los participantes propusieran un problema libre de diseño, y en este contexto, también que entendieran una nueva manera de diseñar: escribir sobre el dibujar. En muchos casos, la selección de estas fue empírica, sin un conocimiento mínimo del problema, alentado por la curiosidad de poner a prueba si estas tecnologías podían resolverlo, un problema que ya habíamos identificado en otros talleres.

Grupo X. En el año 2009, se analizó un Grupo en Chile que usó *RS* con edades de 24 a 27 años, con el fin de comparar la metodología utilizada por el grupo anterior, por otra que no proponía problemáticas libres, sino un problema propuesto por el instructor. Este se centraba en un conjunto de ejercicios basados en una malla tridimensional dentro de la que se distribuían superficies o sólidos. Este método fue seguido paso a paso por los estudiantes. De ese grupo, varios estudiantes comprendieron que la complejidad de una forma o espacio puede ser controlada matemáticamente usando *RS*. Para los estudiantes, esta base fue una motivación para programar, pero decidieron no seguir usando la programación escrita por encontrar en ella un largo camino de aprendizaje y al mismo tiempo, la presentación del resultado no es instantánea. Eligieron así, la programación visual con *GH*. Uno de ellos, empezó su auto aprendizaje, con la traducción al español del Manual Oficial de *GH* editado por Andy Payne, para luego fundar un taller de enseñanza como una plataforma para exponer aquello que han venido aprendiendo.

Los casos más representativos de *A* y *X*, han ido más allá de la experiencia en el taller, porque tuvieron una motivación personal, y un aprendizaje del proceso, que son el instrumento para seguir mejorando. En ambos casos, la reutilización de código fue el punto de partida, porque los instructores recomendaban uno u otro *script* existente (*Grupo A*), mientras que para el *Grupo X* el punto de partida, fue la experiencia del instructor en un determinado método producido por el mismo, enseñando diferentes exploraciones con las variaciones del código.

Grupo B. En el año 2010, se implementó este Grupo, con alumnos de pre-grado y de sexto nivel, en un curso de un semestre (48 horas) en el que se impartió *Rhino* y *GH*, ambos dentro de una asignatura paralela al taller de diseño. La implementación con *Rhino* no tuvo problemas, pero para *GH*, se detectaron dos tipos de situaciones. Los participantes carecían de una base lógico-matemática que les permitiera entender las relaciones entre variables y componentes, repitiendo los diagramas y grupos, pero sin comprender que la solución iba a una problemática particular y que esa solución no podía generalizarse a cualquier problema, agregando que no

estaban en capacidad de definir una problemática de diseño. Es así que se utilizaron ejercicios guiados, con el fin de conseguir un aprendizaje basado en casos de estudio.

Grupo C. En el año 2011, se retiró el uso de *GH*, y *Rhino* pasó a ocupar todo el semestre, pero esta vez ubicando el curso en el curto nivel de estudios. En el 2012, se incluyó el proceso la fabricación. Los estudiantes construían un modelo de su taller o una forma compleja representativa de la arquitectura contemporánea y finalmente la fabricaban. Ello permitió comprender, que no sólo la representación era un medio, sino que *Rhino* facilitaba las piezas, así como la numeración de cada parte para su ensamble y construcción.

Grupo Y. Aparece al mismo tiempo de creado el Grupo C, con estudiantes egresados de edad promedio de 22 a 35 años, quienes en el lapso de un trimestre (72 horas) llevaron *Rhino* (48 horas) y *GH* (24 horas) con el fin de familiarizarse con la programación visual. Sólo dos de los once participantes estuvieron motivados a presentar una solución a una problemática personal luego de terminado el taller, reutilizando y modificando otros códigos como punto de partida. El uso de *GH*, ha sido sostenible en aquellos alumnos que lo han incluido en su trabajo diario, sea en taller o en su vida profesional. No sólo han logrado resultados que sorprenden a los propios autores, sino que el recurso los introduce en una cultura de procesos personalizados. Para Ricard (1982, 20) un objeto es más complejo y tiene mayores posibilidades de potenciarse, cuanto mayor sea el nivel intelectual de quien lo produce.

Conclusiones y trabajo futuro

Latinoamérica ha experimentado una disminución de usuarios en *RS* entre el 2008 y el 2012, como se puede ver en la Fig. 1., diferente a porcentajes estables en otras regiones.

Leitao et al. (2012, 160) sostienen que sus estudiantes prefieren *GH* sobre *RS*, por ser este último considerado un lenguaje obsoleto; al mismo tiempo Celani et al. (2012) encontraron que sus estudiantes optaron por usar *GH* sobre AutoLISP. En ambos casos, la razón fue que les tomó a sus estudiantes más tiempo y esfuerzo aprender la programación escrita sobre la visual. Pero ambos estudios también concluyeron que aunque es largo el proceso de aprendizaje e implementación de la programación escrita, este se recupera cuando el estudiante se enfrenta a problemas más complejos no sólo de forma y espacio, sino de gestión del problema.

Aunque la preferencia por *GH* y la creación de código en este entorno se dan con mayor facilidad según estas experiencias, la modificación por terceros no lo es. Davis et al. (2011, 363-364) descubrieron en una revisión de 1982 definiciones producidas por 575 usuarios con *GH*, que servían muy poco para ser utilizadas por otros: un 56% de ellas no incluyó el nombre de sus parámetros, lo que hacía difícil entender como estaba estructurado y en un 97.5% los componentes no estaban anidados o agrupados. Así, la secuencia de relaciones y operaciones para automatizar la construcción de una geometría se hacían difíciles de editar o revisar. Esto no ocurre con la programación escrita usando *RS*, porque la secuencia es lineal y hay una estructura permanente de grupos de datos, y aunque un diagrama en *GH*

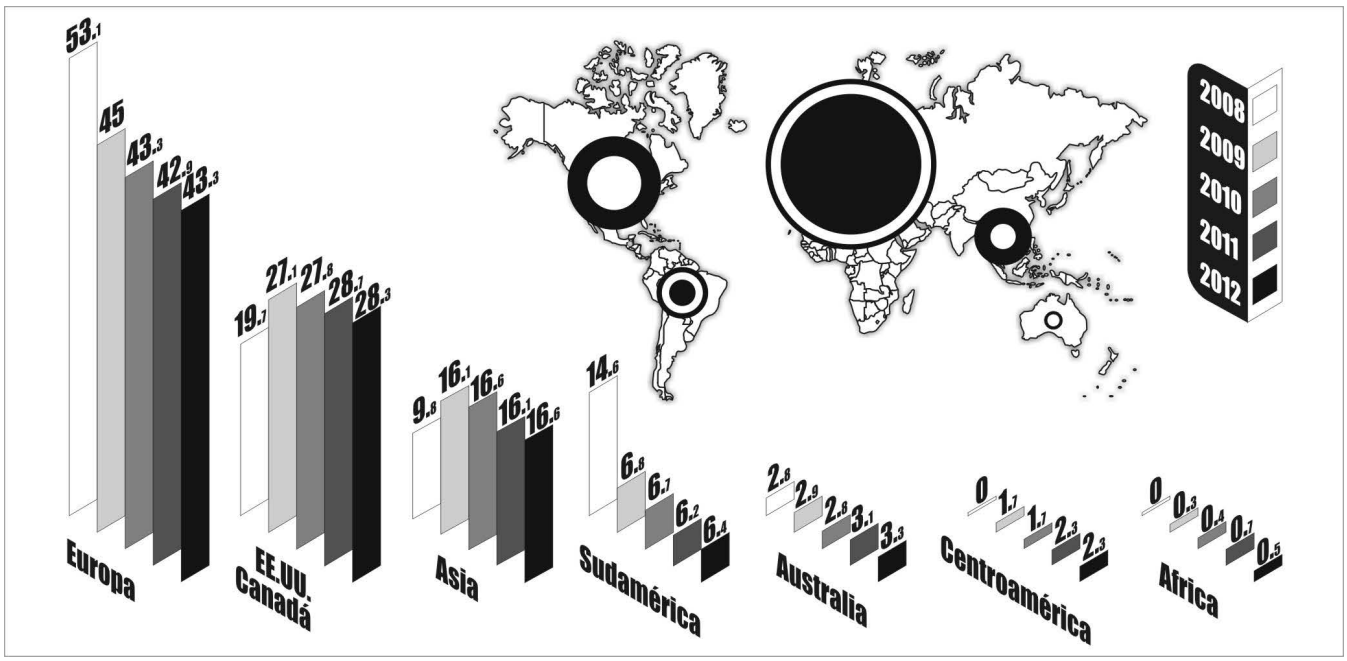

Fig. 1. Distribución geográfica de usuarios de Rhinoscripting en base a estadísticas de Motigo entre el 2008 y el 2012 provenientes del blog http://rhinoscriptingresources.blogspot.com

facilita construcción, ésta sólo es entendida por quien la produce. Los *EUPs* más experimentados, prefieren utilizar *RS* porque algunos algoritmos son demasiado complejos para *GH*. A pesar de ello, el uso de diagramas para visualizar un proceso, será una herramienta de uso corriente para los arquitectos. La programación visual, es la versión personalizada del software interactivo, pero la complejidad de un problema pone límite a esta libertad. Aunque permite programar dentro de cada componente, no todos estarán interesados en utilizarla, como sucedió con la programación escrita, estamos acostumbrados a modificar una solución, en vez de programarla por nuestra cuenta. La cultura automatizada y modificación de bloques e información fija propia de las primeras generaciones de CAD ha dejado una cultura que vemos reflejada en nuestra preferencia por la interfaz interactiva, información repetitiva que no produce conocimiento en un proceso. Probablemente, la programación visual en el futuro, será la plataforma de crítica de un diseño y aunque su naturaleza nos permite tener un resultado único por la amplia cantidad de variaciones que podemos hacer, el procedimiento y naturaleza de la solución seguirá siendo la misma. Los *waffles* y *atractores* utilizados en el entorno de *GH*, se convertirán en las cajas y esferas de la arquitectura contemporánea. Los casos de estudio expuestos, son en su conjunto, una referencia de problemas dentro de una implementación ambiciosa que hasta el momento, no ha generado resultados masivos en Latinoamérica. Sin embargo, ha despertado curiosidad, saber que un proceso puede ahora ser utilizado por otros, para crear nuevas soluciones o para combinar posibilidades de acuerdo a situaciones nuevas. Los bloques de CAD han sido reemplazados por los componentes de un proceso.

Referencias

Andersen, P., Bennedsen, J., Brandorff, S., Caspersen, M., Mosegaard, J. 2003. Teaching programming to liberal arts students: a narrative media approach. *ACM SIGCS,* 35 (3), 109-113.

Burry, M. (2011). *Scripting Cultures: Architectural Design and Programming*. Sussex: John Wiley and Sons.

Celani, G., Verzola, E. 2012. CAD Scripting And Visual Programming Languages For Implementing Computational Design Concepts: A Comparision From A Pedagogical Point of View. *International Journal of Architectural Computing*, 10 (1), 121-137.

Converso, S. 2008. *SHoP Works: Digital Constructive Collaborations*. Roma: Edil Stampa.

Davis, D., Burry, J., Burry, M. 2012. Understanding visual scripts: Improving collaboration through modular programming. *International Journal of Architectural Computing*, 9 (4), 361-375.

Harrison, W. 2004. From the Editor: The Dangers of End-User Programming. *Software IEEE*, 21 (4), 5- 7.

Herrera, P. 2011. *Rhinoscripting y Grasshopper a través de sus instructores: un estudio de patrones y usos* (pp. 180-183). Santa Fe: Universidad Nacional del Litoral.

Kedan, E. 2012. *Provisional: Emerging Modes of Architectural Practice USA*. New York: Princeton Architectural Press.

Leitao, A., Santos L., Lopes, J. 2012. Programming Languages For Generative Design: A comparative Study. *International Journal of Architectural Computing*, 10 (1), 139-162.

Loukissas, Y., Sass, L. 2004. Rulebuilding: 3D Printing: Operators, Constraints, Scripts. (pp. 176-185). Ontario: University of Waterloo.

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., et al. 2008. Debugging: A review of the literature from an educational perspective. *Computer Science Education*, 18 (2), 93-116.

Mitchell, W. 2010. *Foreward*. En R. Krawczyk (Ed.), *The Codewriting Workbook: Creating Computational Architecture in Autolisp* (pp. 7-8). New York: Princeton Architectural Press.

Nardi, B. 1993. *A small Matter of Programming*. Cambridge, MA: MIT Press.

Lyon, A. 2007. Foro IV: Epistolar. *Revista de arquitectura*. 15 (1), 8-13.

Reas, C., McWilliams, Ch., Barendsen, J. 2010. *Form+Code in Design, Art, and Architecture.* New Jersey: Princeton Architectural Press

Robins, A., Rountree, J., Rountree, N. (2003) Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13 (2), 137-172.