Patrones en la Enseñanza de la Programación en Arquitectura: De la Hetero-Educación a la Auto-Educación en Latinoamérica

Patterns in the Teaching of Visual Programming in Architecture: From the Hetero-Education to Self-Education in Latin America

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

Abstract

Teaching programming to architects, in academic and professional contexts, occurs in Latin America through self-management, and focused on results, without analyzing the processes and establishing a follow-up to participants, to establish patterns of application. The pointing out of these problems and the proposal of how to make said education sustainable has allowed finding variables specific to the region and to the very same tools and instruments, which are constantly evolving. At the same time, it is proposed after the analysis, that hetero-education (shared learning) itself requires self-education (self-teaching processes) as a complementary process.

Keywords: Visual programming; Scripting; Hetero education; Self education; Rhino.

Introducción

Castells (2009) sostiene que la economía actual depende de dos tipos de trabajadores, los genéricos y los auto-programables. "Los primeros trabajan por instrucciones y pierden a lo largo del tiempo su capacidad de negociación y valor, a diferencia de los que adaptan sus capacidades al ritmo cambiante de la tecnología" (p. 57-60). Se puede inferir que una equivalencia en los procesos informáticos la propuso Terzidis (2006). Así, *computarizado* es lo pre-programado o genérico, se refiere a la automatización y mecanización. C*omputacional* permite la exploración indeterminada y cambiante, se trata de razonamiento, lógica, algoritmo y auto-programación. También, el medio digital que usamos puede ser una herramienta o un instrumento. Converso (2012) define una *herramienta* como algo "que brinda siempre el mismo resultado" y al *instrumento* "que permite la exploración" (p. 21). En los casos de estudio investigados, los participantes aspiraban a ser auto-programables, apropiarse de tecnologías computacionales y manipular un instrumento en su trabajo diario.

Pero Castells (2009) también advirtió, que "la inmensa mayoría de los trabajadores del planeta y la mayoría de los países avanzados se sigue constituyendo mano de obra genérica" (p. 58). Rushkoff (2010) sostiene que la programación es punto clave en una sociedad digital. "Si no aprendemos a programar, nos arriesgamos a ser programados" (p. 133). Pero, no todos estamos dispuestos a ser programadores. Burry (2011) estudiando a programadores en arquitectura, los define como "diseñadores nacidos o con disposición para programar" (p. 34). Están dispuestos a aprender por mismos, como lo afirmó Senske en el 2005, en su tesis "Fear of Code: Approach to integrating computation with architectural design" al también analizar a estudiantes de arquitectura que programaban con Rhinoscripting. También Burry (2011) afirma que "son muchos los creadores de código que se están clonando, amparados por la bandera de la legitimidad contemporánea" (p. 50). Es decir, la exposición del código ha permitido que otros usuarios reproduzcan y reutilicen algoritmos, adaptándolos en algunos casos a desarrollos propios (especialmente en estudios de maestría y doctorado), pero también se han convertido en sólo ejercicios de exploración que se dejaron en el olvido (particularmente en talleres breves). A pesar que la programación en Latinoamérica se inició hacia el 2006, sus resultados han producido muy poco efecto en el campo profesional. Siendo el espacio académico, el que la adoptado como referente tecnológico (Herrera, 2011). En este contexto, se propuso hacer un estudio preliminar para entender cuál era la situación de los participantes de talleres breves luego de haberlos introducido en ejercicios de representación y programación.

Este estudio se inició, con estadísticas que afirman que la programación escrita no ha llegado a ser parte de implementaciones sostenibles en la región como si lo ha sido la programación visual, que fue promovida por varios factores que han sido tratados anteriormente por Celani & Verzola (2012) y Herrera (2012).

Desde la década de 1990, la implementación de tecnologías digitales en nuestra región, se produce con software por instrucciones a intentos de programar por nosotros mismos. La problemática se presenta, cuando el participante espera sólo

recibir un conjunto de instrucciones que le sirvan más adelante como una receta paso a paso en la solución de un problema, como sucede en la enseñanza instructiva de AutoCAD, 3DS Max, Revit o ArchiCAD. En ese contexto, la enseñanza tradicional de técnicas digitales por instrucciones no es la misma cuando se trata de programación, especialmente a largo plazo, como se podrá revisar en este estudio.

Ivan Illich (2005, p. 201-204) citado por Braham & Moe (2011, p.4), "sostuvo que la moderna era tecnológica, que se caracteriza por herramientas, instrumentalidad y función, dio paso a finales del siglo XX a la era de los sistemas, que se caracterizan por complejas configuraciones, auto-organización y emergencia. (…) el cambio de herramientas a sistemas, demanda a la comunidad académica proponer soluciones híbridas." Emmons (2012) propuso que "este es uno de los desafíos de la siguiente generación de educadores" (p. 304)

Touriñan (1996), sostiene que en la auto-educación "el educando puede ser agente del cambio educativo, de la propuesta de cambio y de los modos de lograrlo", por el contrario, en la hetero-educación "el educando es agente del cambio porque hace lo que le proponen. Pero no es por sí solo el agente de la propuesta del cambio y de los modos de lograrlo, porque hay otro agente, el educador" (p. 70). El intercambio de información entre el instructor y el estudiante no es suficiente para asegurar que lo enseñado produzca cambios significativos al emplear la programación en tareas que antes se resolvían con técnicas de dibujo.

La auto-educación, es uno de los patrones de implementación en programación visual y escrita, que se motiva por la curiosidad: ¿cómo lo hago? o ¿cómo lo copio? y por el reto ¿podré hacerlo? o ¿podré aprenderlo?. Manovich (2005) lo describe como un proceso que se experimenta como una narración y que "a medida que se avanza, se va descubriendo poco a poco, las reglas que operan el universo construido por ese juego. Se aprende su lógica oculta; en definitiva, su algoritmo" (p. 288).

Desde el 2008, se han publicado experiencias de la región sobre la enseñanza de programación con procesos de hetero-educación. Pero a diferencia de otras implementaciones, centradas sólo en programación, se propuso una enseñanza híbrida usando herramienta e instrumento, con técnicas de representación a través de Rhinoceros y razonamiento para solucionar un problema con programación visual usando Grasshopper.

Metodología

La auto-educación, promovida desde la hetero-educación (ambas dentro de un espacio educativo no formal, es decir fuera de un espacio universitario), tiene un conjunto de patrones de adaptación que ha sido poco estudiado. Partiendo del control que debemos tener de un problema, mientras más cercano estemos de entender las técnicas de un software, nos acercaremos más a lo que propuso desde un principio su programador. Manovich (2005) afirma, que el "software, contiene un conjunto de convenciones programadas, que guardan en su organización, una gran cantidad de posibles soluciones, a las que no podemos acceder como un catálogo, sino sólo de acuerdo a nuestra capacidad de selección y combinación" (p.182). La selección y combinación de técnicas y casos de estudio, fue realizada por los instructores y propuesta a los participantes. Los participantes desarrollaron ejercicios de representación para entender las técnicas de modelado de formas complejas, que era la base para explorar otras soluciones a problemas con programación. Esto, porque la tradición de enseñar software como una herramienta, se promueve para controlar una solución que está casi resuelta por el diseñador y que sólo es necesario representarla. Por lo tanto, el participante no se enfrenta a una situación nueva como lo haría al programar por primera vez.

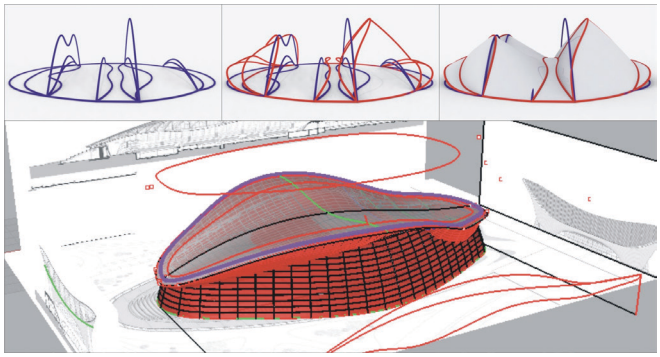

Figura 1: Casos de estudio: Pabellón UEA de Norman Foster y el Aquatic Center de Zaha Hadid. Modelo desarrollado por los participantes en horas no lectivas.

Bajo esta premisa, se desarrolló en Lima, Perú, la primera parte del taller, con una selección de técnicas destinadas a despertar en el participante el control de la forma bajo una secuencia consecutiva de instrucciones. El participante, se involucró con diferentes herramientas para proponer una solución en la que la variabilidad de su propuesta era mínima.

Se observó y analizó entrevistas y encuestas como parte del seguimiento a estos talleres breves a lo largo de un año, Cada taller de tres, duró tres meses. Estuvo bajo la tutela de dos instructores con un máximo de 20 participantes. A diferencia de los talleres de programación visual destinados a grupos de edad homogénea, se propuso tener grupos de control de diferentes edades. Los participantes eran estudiantes, egresados y profesionales en ejercicio de ambos sexos, con edades variables entre 22 y 55 años. Los problemas que se exploran en este estudio, se contrastaron con el análisis de otros problemas encontrados en talleres breves de programación escrita y visual documentados entre el 2006 y 2012. Los talleres, se realizaron con casos de estudio, implementando progresivamente diferentes niveles de complejidad por cada sesión en un primer periodo teórico/práctico de 48 horas. En la segunda parte se utilizó Grasshopper, en un curso introductorio a la programación visual, potenciando las técnicas previas, y que los preparó para darle continuidad a la etapa de aprendizaje autónomo.

Figura 2: Caso de estudio: Estadio Hangzhou por NBBJ. Modelado con Rhino y parametrizado en alguna de sus partes con Grasshopper. Modelo desarrollado en clase por los participantes.

Resultados

Patrones de Enseñanza

Desde que la programación escrita y visual se experimenta en Latinoamérica se han identificado patrones de implementación propuestos por iniciativas autogestionarias y otras del tipo sistematizado (Herrera, 2011, p.179-182). Así mismo, los patrones y convenciones utilizados en la enseñanza de programación escrita con *Rhinoscripting* (Herrera, 2009, p. 341) han sido también utilizados en la enseñanza de la programación visual con *Grasshopper*.

En general, estas implementaciones fueron empíricas, sin llegar a ser masivas. Se dirigieron a motivar y acercar tecnologías emergentes a la región. Son implementaciones que se promovieron desde la academia y no desde la profesión. Fue realizada por grupos que venían estudiando maestrías y doctorados en EE.UU. o Europa. La enseñanza fue del tipo hetero-educacional, y se adaptó en algunas universidades con patrones similares a otros talleres breves realizados en diferentes partes del mundo. Modelos similares a estas experiencias de enseñanza, se presentaron en la exhibición inglesa: *Spontaneous Schooling*, realizada en junio del año 2010, donde se mostraron 86 experiencias de diversas nacionalidades. Estos trabajos incluyeron resultados académicos y profesionales en programación visual y escrita como instrumentos para diseñar y fabricar. Atwood, (2011, p.160) los describe como "una exploración del rol de los talleres breves o *workshops*, dentro de las disciplinas de diseño". La curadora de la exhibición, Melissa Woolford explica que en los cursos universitarios de un semestre, los fracasos son controlados o no están permitidos, mientras que en los talleres de breves, el fracaso se contempla como una posibilidad, aprendiendo de una metodología y del proceso.

En la implementación de nuestra región, los patrones utilizados en talleres breves ha sido de dos tipos: a) Potenciando la idea: cada grupo, propone un problema y programan sobre sus problemas y b) Potenciando el código: todos entienden un problema y programan sobre el mismo problema (Herrera, 2012). Al seguir a estos estudiantes se encontró poca o nula aplicación en su vida diaria. Por ello, se propuso para este estudio, una metodología distinta, que potenciara la representación de un problema complejo, el entendimiento de la naturaleza de un pseudocódigo y el análisis de las soluciones explorando códigos. Antes de iniciar, se identificaron los problemas que podrían limitarlo.

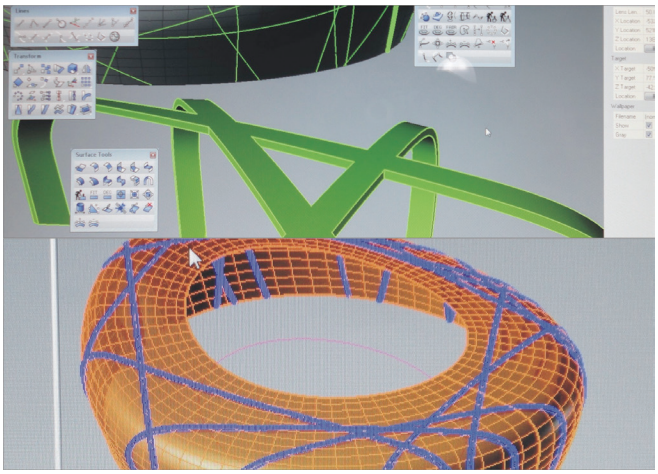

Figura 3: Caso de estudio: Beijing National Stadium de Herzog & De Meuron. Modelo desarrollado en clase por los participantes.

Problemas Antes de la Implementación

La enseñanza universitaria para no egresados, así como en su modalidad de cursos libres para egresados, se centra en ofrecer determinadas técnicas digitales que le servirán al participante en un futuro profesional cercano. Sin embargo, el contexto en donde se espera aplicar lo aprendido, es ajeno al uso de tecnologías emergentes, por lo tanto su uso se realiza a largo plazo o nunca. No sólo sucede con la programación sino también con procesos de gestión como la implementación del Building Information Modeling (BIM). El 99% de los participantes nunca había usado Rhino, pero había escuchado de él. Al empezar, tampoco conocían la diferencia entre Rhino y Grasshopper. El 100% era usuario de AutoCAD y suponía que aprendería un grupo de técnicas que utilizaría como arquitecto para resolver algún problema de representación. Al terminar cada taller, se hizo un seguimiento mediante encuesta, y se encontró algunas limitaciones luego de la implementación: problemas del tipo personal (parámetros de diseño pre-establecidos y limitaciones generacionales, especialmente en los participantes de más de 40 años), de clientes

y usuario del servicio (demasiado conservadores, y que consideran que a mayor tecnología, mayor costo del proyecto), y laboral (al no ser común en los proyectos, es difícil darle continuidad a procesos más complejos y reales). Este efecto, produce una cantidad limitada de usuarios que estén usando un instrumento para la exploración de soluciones a problemas de diseño.

Tanto en la programación escrita como en la visual, se requiere conocer las técnicas que se utilizan en Rhino. En las primeras versiones del taller, la implementación se realizaba asumiendo que si el participante conocía las técnicas de modelado de formas complejas usando Rhino, podría eventualmente integrar fácilmente los componentes de Grasshopper en su vocabulario cotidiano. Pero hasta que no fue incluido como requisito y enseñado por nosotros dentro de los talleres breves, no se pudo asegurar, que todos los participantes del proceso, tenían un nivel de conocimiento que les permitiera pasar de la imitación de técnicas pre-programadas a programadas.

Problemas Después de la Implementación

Patrones para reutilizar sus códigos

La enseñanza de programación como sucede en otras especialidades que lo emplean, deja abierto el código del proceso y la variabilidad del resultado, para poder regresar a él cuantas veces sea necesario. Pero esta información con el tiempo deja de ser útil, si no está estructurada. Davis (2013, p.136-142) hizo una lista de elementos clave que podrían guiar la comprensión del participante cuando produce código, utilizando en sus definiciones: Nombres, Posicionamiento, Explicaciones y Agrupaciones. Davis (p.130) también encontró al analizar 2002 definiciones publicadas en el repositorio de Grasshoper3D entre mayo del 2009 y agosto del 2011 lo siguiente: que sólo el 36% de los modelos tenía al menos una pieza de texto que explique alguna parte del código, en un 30% de las definiciones se usó uno o más grupos. Sólo el 19% tenía algún nodo con nombre. Sólo el 2.5% empleó módulos, 48% no sólo no empleó módulos, sino tampoco agrupaciones, explicación de lo que hicieron, ni nombres de los parámetros. Este repositorio, al ser un foro y un espacio de intercambio de información, no está organizado en función a métodos, sino a entradas con problemas particulares. En su conjunto, estos problemas evidencian que aunque los usuarios puedan acceder a muchos códigos, no están estructurados ni organizados, lo que en muchos casos no permite entender lo que hizo el autor original. Esto se comprobó al solicitar a los participantes a una semana de finalizar las sesiones de Grasshopper, que se familiarizaran con un método e investigaran sobre las variaciones de varios autores, sobre el mismo problema. Los participantes encontraron muchas dificultades para entender componentes que no les eran familiares, si la definición era muy antigua, entonces las versiones no eran compatibles. El 1° de octubre del 2013, David Rutten (2013), administrador y creador de Grasshopper y el foro del mismo nombre publicó una entrada bajo el título “How to get help when you have a problem". El mayor foro para esta aplicación, se ha convertido con los años en un espacio de soporte en línea a problemas que en su mayoría no son situaciones nuevas, sino problemas de adaptación. La permanente actualización y corrección de las versiones de Grasshopper, genera muchas preguntas sobre componentes encontrados en definiciones descargadas, pero que ya no existen, como por ejemplo “Receiver” que desde el 2010 esta anidado en todos los componentes. Pero aún se siguen recibiendo preguntas de donde encontrarlo. A ello se añade, que desde la aparición del primer manual en el 2009 escrito por Andy Payne, no ha aparecido otra guía que muestre las modificaciones realizadas en cada versión.

Patrones para seguir entendiendo códigos

En los talleres breves que se desarrollaron para este estudio se propusieron dos grupos de control. El primer grupo, usó patrones convencionales de anotación escrita a mano. Estos participantes escribían y dibujaban los diagramas de cada sesión. Luego de unas semanas, al intentar regresar a repasar la información, notaron que no podían deducir el origen de algunas instrucciones, porque en muchos casos alguna parte del proceso se memorizó, confiando en que al verlo anotado, podrían recordar su origen. Un segundo grupo fue instruido para utilizar la captura de pantalla. A diferencia del video en el que no se puede escribir durante el seguimiento de las instrucciones, en este caso, los participantes guardaban las imágenes en un procesador de texto y anotaban detalles de cada proceso. Al principio, los participantes, dudaron en que esta manera de anotar, les sirviera. Pero sin darse cuenta, tenían casi un manual de sus clases, lo que les permitió meses después de haber terminado el taller, poder revisar y utilizar nuevamente esta información. Así mismo, se instruyó a los participantes a agrupar, nombrar grupos y componentes, así como a usar notas y explicaciones en sus definiciones.

Experimentados y novatos

Talbot (2004) afirma que en general los participantes experimentados “aportan a sus estudios técnicas, conocimientos, opiniones e ideas pertinentes” (p.37). Por otro lado, en diseño, Ricard (1982) sostuvo que “un objeto es más complejo y tiene mayores posibilidades de potenciarse, cuanto mayor sea el nivel intelectual de quien lo produce” (p.20). En la Figura 4, se muestra el pseudocódigo: Centro, Polígono, mover puntos pares e impares, mover el centro, trazar líneas de puntos hacia el centro, dividir líneas pares e impares, trazar líneas entre puntos. Deducido desde un objeto real que conocían los participantes.

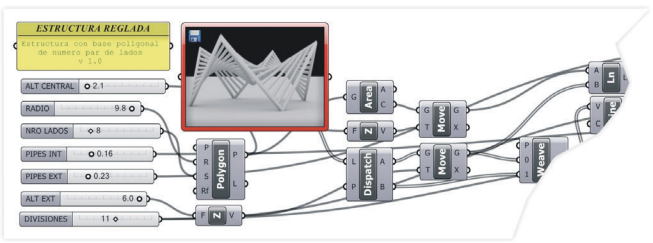

Figura 4: Estructura Paraboloide. Modelo desarrollado en clase por los participantes.

La ventaja de tener un grupo de estudio con participantes nacidos entre 1960 y 1972 aporta significativamente en el entendimiento de una problemática, ayudándolos a definir con facilidad el pseudocódigo, a diferencia de los participantes nacidos entre 1980 y 1990.

Conclusiones

Este es un trabajo en progreso, que se propone luego de esta revisión, articular un modelo de enseñanza híbrido que se inserte gradualmente y de manera compartida entre una enseñanza apoyada por un instructor pero potenciada por el mismo participante, es decir, buscar que se establezca un aprendizaje autónomo sistematizado empezando por anticipar los problemas descritos. Los resultados de las encuestas demuestran lo significativo que fue el avance progresivo y gradual desde Rhino hasta Grasshopper y se comprobó que el uso de situaciones familiares a los estudiantes, despiertan en el participante una motivación que los conduce a indagar más soluciones para otras partes del mismo caso de estudio, es decir, no abordarlos con muchos problemas, sino seleccionar problemas específicos complejos y abordarlos desde herramientas pre-programadas e instrumentos auto-programables.

Los más jóvenes mostraron predisposición para adaptarse a los métodos y procedimientos mientras que los más experimentados necesitaron más tiempo para asimilarlos. Pero relacionando ambos grupos en un espacio común, se fortalecieron ambos. Dejando evidencia que debería incluirse profesores o asistentes de taller, junto a alumnos, bajo la modalidad de andamiaje o taller vertical.

Los resultados, permiten pasar a otra etapa de implementación de manera sistemática, con casos de estudio que se adapten a problemáticas más cercanas a los participantes, según los requerimientos de cada grupo.

Aún no se puede afirmar cuanto cambiará Grasshopper en los siguientes años, ha sido bastante desde su primera versión en el 2008. Lo que sí se puede anticipar, es que entender problemas con el computador como aliado de nuestros procesos, nos dará ventajas, sin importar la plataforma disponible en ese momento.

Agradecimientos

A Pedro Arteaga por la confianza depositada en este proyecto, acompañándome en cada implementación. Y todos los estudiantes que forman parte de esta implementación.

Referencias

Atwood, B. (2011). Spontaneous Schooling. Journal of Architectural Education. 64(2), 159-161

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

Castells, M. (2009). Comunicación y Poder. Madrid: Alianza Editorial

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. (2012). Shop Works. Digital Constructive Collaborations. Basel: Birkhauser

Davis, D. (2013). Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture. Tesis de Doctor en Filosofía. School of Architecture. RMIT University.

Emmons, P. (2012). Drawing and Representation. The Uncertain Future of Craft: From Tools to Systems. J. Ockman (Ed). Architecture School: Three Centuries of Educating Architects in North America.

Herrera, P. (2009). Patrones y Convenciones en el uso de Rhinoscripting. E. Nardelli & Ch. Vincent (Eds.), Libro de Ponencias SIGraDi 2009 (pp. 340-342). Sao Paulo: Universidade Prebiteriana Mackenzie

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

Herrera, P. (2012). Reutilizando códigos en arquitectura como mecanismo de información y conocimiento. De la programación escrita a la visual. D. Cardoso (Ed.), Libro de Ponencias SIGraDi 2012 (pp. 74-78). Fortaleza: Universidade Federal do Ceará.

Ivan I. (2005). From Tools to Systems. D. Caley (Ed). The Rivers North of the Future: The testament of Ivan Illich. Toronto: House of Anansi Press, Inc.

Manovich, L. (2005). El Lenguaje de los nuevos medios de comunicación. La imagen en la era digital. Barcelona: Paidos.

Braham, W. & Moe, K. (2011). Performative Practices: Architecture and Engineering in the Twenty-First Century. Washington: Association of Collegiate Schools of Architecture.

Ricard, A. (1982). Diseño ¿Por qué?. Barcelona: Editorial Gustavo Gili.

Rutten, D. (2013). How to get help when you have a problem. Retrieve from http://www.grasshopper3d.com/forum/topics/how-to-get-help-when-you-have-a-problem.

Rushkoff, D. (2010). Program or Be Programmed: Ten Commands for a Digital Age. Berkeley: Soft Skull Press

Talbot, C. (2004). Estudiar a Distancia. Una guía para estudiantes. Barcelona: Gedisa.

Terzidis, K. (2006). Algorithmic Architecture. Oxford: Architectural Press.

Touriñan (1996). Análisis conceptual de los procesos educativos. Formales, no formales e informales. Teoría de la Educación. N°8, pp. 55-80