La programación de ordenadores. Reflexiones sobre la necesidad de un abordaje interdisciplinar *

A programação de computadores. Reflexões sobre a necessidade de uma abordagem interdisciplinar

Computer Programming. Reflections on the Need for an Interdisciplinary Approach

Verónica S. D'Angelo **

La escisión entre ciencias sociales y formales, entre filosofía y tecnología, entre máquina y ser humano, en una sociedad que subestima el saber técnico pero se sirve de él, ha promovido equívocos y sesgos que condicionan el lugar asignado implícitamente a los profesionales del *software* y dificultan el diálogo interdisciplinar. Este artículo se propone revisar los fundamentos de la ciencia de la computación, y de la programación en particular, con una mirada retrospectiva sobre ciertas experiencias en la universidad y prospectiva hacia las nuevas propuestas de enseñar programación en las escuelas, apelando a una participación más activa de la filosofía y la psicología que permita distinguir entre intereses educativos e intereses de mercado.

Palabras clave: programación de ordenadores; filosofía; psicología; tecnología; interdisciplina; ciencia de la computación; programación en las escuelas

---

* Recepción del artículo: 26/11/2016. Entrega de la evaluación final: 18/05/2017.
** Analista de sistemas de información, docente universitaria, Magíster en psicología cognitiva y aprendizaje FLACSO/Universidad Autónoma de Madrid (en curso), España. Correo electrónico: catudan@arnet.com.ar.

A cisão entre ciências sociais e formais, entre filosofia e tecnologia, entre máquina e ser humano, em uma sociedade que subestima o conhecimento técnico, mas faz uso dele, promoveu desentendimentos e distorções que condicionam o lugar implicitamente atribuído aos profissionais de *software* e dificultam o diálogo interdisciplinar. Este artigo tem o objetivo de revisar os fundamentos da ciência da computação, e da programação em particular, com um olhar retrospectivo sobre determinadas experiências na universidade e prospectiva para as novas propostas de ensino de programação nas escolas, apelando a uma participação mais ativa da filosofia e psicologia que permita distinguir entre interesses educacionais e interesses de mercado.

Palavras-chave: programação de computadores; filosofia; psicologia; tecnologia; interdisciplina; ciência da computação; programação nas escolas

The split between social and formal sciences, between philosophy and technology, between man and machine, in a society that underestimates technical knowledge at the same time that it takes advantage of it, has fostered misunderstandings and biases that constrain the place implicitly assigned to software professionals and hinder interdisciplinary dialogue. This paper intends to review the fundamentals of computer science, and of programming in particular, with a retrospective look on certain university experiences and a prospective one towards the new proposals for teaching programming in schools, calling for a more active participation from philosophy and psychology, in order to allow a better distinction between education and market interests.

Keywords: *computer programming; philosophy; psychology; technology; interdiscipline; computer science; programming in schools*

1. Una dificultad del Sys Analysis

Alguien que no ve con simpatía suficiente una cosa,
no la comprenderá tampoco fácilmente.
Sigmund Freud, "Una dificultad del psicoanálisis" (1917)

Sys Analysis es la abreviatura en inglés de Systems Analysis que refiere al análisis de sistemas de información. La dificultad mencionada proviene de tres cuestiones a considerar. En primer lugar, afirmamos que la programación de ordenadores no debiera concebirse escindida del análisis de los requerimientos del sistema de información que se desea automatizar. La aparente obviedad de dicha afirmación no es tal si consideramos que la programación está adquiriendo un carácter masivo al ser ejercida por profesionales, aficionados y estudiantes cada vez más jóvenes, en contextos que distorsionan u omiten la relación entre análisis (representación del problema) y programación (codificación). Entendemos que, aunque el rol del analista ha desaparecido casi totalmente de las organizaciones, quedando subsumido en el nuevo rol del desarrollador, las actividades de análisis y diseño en tecnología de la información (TI) continúan realizándose (Baker, 2014; Disciplined Agile Consortium, 2017; Eriksson, 2017; Pichler, 2010).

En segundo lugar, existe una manera tradicional e intuitiva de entender la computación como "la disciplina que trata sobre las computadoras". En el surgimiento de un campo disciplinar, es fácil confundir la esencia del campo con sus herramientas. Una buena analogía es el nacimiento de la geometría, que para los egipcios trataba sobre instrumentos de medición; sin embargo, visto en retrospectiva, lo que hacían era formalizar las nociones de espacio y tiempo que más tarde conducirían a métodos axiomáticos (Abelson, 2005).

En tercer lugar, se deriva de lo anterior que los analistas de sistemas son percibidos como los profesionales que se ocupan de las computadoras. Según la visión tradicional, los analistas son percibidos como técnicos, al igual que otros cientistas de la computación (Fellows, 1993). Esta definición ya no es viable porque el análisis de sistemas o el denominado pensamiento computacional no sólo ha sido aplicado a los sistemas artificiales, sino también a los naturales, quedando en evidencia que la computadora es sólo una herramienta, pero la computación es el principio (Denning, 2010; Denning, 2017; Baltimore, 2002). A lo largo de la historia, las sociedades siempre han tenido necesidad de procesar información, por lo que el nacimiento de la computación precedió a la invención de la computadora (Aspray, 1990).

La verdadera ciencia de la computación (CC) no trata sobre las computadoras, así como tampoco la astronomía trata sobre los telescopios, ni la biología sobre los microscopios, porque ninguna ciencia trata sobre los instrumentos que utiliza (Abelson, Sussman y Sussman, 1996; Denning, 1984; Fellows, 1991; 1993; Knuth, 1975). La computación existe con anterioridad a los artefactos que permitieron facilitarla e incluso distorsionar su concepto para alimentar una rueda de fabricación

de nuevos productos que no siempre equivalen a nuevas funciones. Confundir el principio con la herramienta conduce en ocasiones a priorizar la utilización de la herramienta en casos en que deberían indagarse los fundamentos, por ejemplo en situaciones de enseñanza-aprendizaje.

La CC como disciplina es el estudio sistemático de procesos algorítmicos que describen y transforman la información: el análisis, el diseño, la implementación, la eficiencia, la aplicación. Para la CC y en particular la TI, la pregunta clave que le da sentido al análisis, al diseño y al cómputo es qué automatizar. Es decir, "¿Qué puede ser (de manera eficiente) automatizado?" (Denning, Comer, Gries y Mulder, 1989).

En la eficiencia se incluye la satisfacción de los usuarios que harán uso del sistema, que no es la computadora. En 1984, Stephen McMenamin y John Palmer realizaron una contribución fundamental a la ciencia de la computación, en particular al análisis estructurado de sistemas que se venía desarrollando desde la década del 70. Con el concepto de "tecnología perfecta" lograron identificar los requisitos verdaderos de un sistema, demostrando que estos no dependen de los límites del *hardware* en el cual se implementan. Del análisis esencial de un sistema se obtiene el conjunto de requisitos de información realmente necesarios, no supeditados a la tecnología física.

En La epistemología de la cibernética, en relación a los sistemas en general, Bateson señala que "las características mentales del sistema son inmanentes, no a alguna de las partes sino al sistema en cuanto totalidad. La computadora es siempre sólo un arco de un circuito más amplio, que siempre incluye un hombre y un ambiente" (Bateson, 1972). Una computadora no puede pensar. Es un sistema cerrado. Pero puede poseer características mentales, vale decir: circuitos autocorrectivos completos. Por ejemplo, un programa que controla iterativamente una variable para decidir si finaliza o continúa con la consulta de datos de alumnado de la Facultad de Psicología necesita que un alumno pulse "Salir" o "Continuar". El sistema, entonces, estará compuesto por el alumno, más la facultad —porque es el ambiente donde el alumno obtuvo datos para ingresar al sistema—, más la parte del programa que controla dicha parte del sistema. "El uso que el hombre le dé a la máquina no es un rasgo de la organización de ésta, sino que es del dominio en que ella opera, y entra en nuestra descripción de la máquina dentro de un contexto más amplio que la máquina misma (Maturana, págs. 67-68). El todo que surge del funcionar de la máquina en contexto es mucho más que sus componentes. Para detectar donde comienza y termina la máquina —el autómata— no es identificar un dispositivo de *hardware*, sino cualquier sistema inteligente que lo incluya.

1.2. Analistas docentes

Con la incorporación de las tecnologías de la información y la comunicación (TIC) a todos los niveles del sistema educativo, quedó en evidencia la percepción que algunos educadores tenían del producto tecnológico: era mayor el beneficio de usarlo que de comprenderlo. Un obstáculo epistemológico para la adopción de una postura crítica ante la tecnología que se ofrecía con el rótulo "educativa" antes de que algunos estudios mostraran que la introducción de la computadora no podía por sí misma

mejorar el aprendizaje (Armstrong y Casement, 2000; Bowers, 2000; Buckingham, 2008; OECD, 2015; Torr, 2003).[1]

El diálogo entre profesionales de la educación y la tecnología consistía en un intercambio utilitario antes que disciplinar; aunque los analistas de sistemas eran bienvenidos para resolver problemas ligados al funcionamiento anómalo de los dispositivos o a la instalación y prueba de *software*, no parecía haber cabida para sus opiniones en cuestiones conceptuales de índole pedagógica, aunque tuvieran formación en ambos campos disciplinares y experiencia como docentes.

Si bien es cierto que la experticia en un campo no conduce per se al conocimiento de su didáctica, un sesgo de proporciones más profundas parecía emerger para excluir a estos profesionales del círculo de confianza educativo: como si algo propio de los técnicos fuera incompatible con el concepto mismo de educación: portadores de un reduccionismo input-output o causa-efecto del modelo mecanicista, cartesiano y energético —casualmente el mismo de la epistemología freudiana—, asociados a las metáforas del procesamiento de la información de una revolución cognitiva confundida con conductismo, al estado burocrático autoritario, al positivismo, al neoliberalismo, a las habilidades superficiales y utilitarias.[2] Dado el paso de transgredir el propio límite disciplinar hacia las ciencias sociales, nuestras aportaciones en jerga escueta nos confinaban a la categoría de oyentes. Tras haber escuchado a un psicólogo afirmar que "un ingeniero es tan sólo una regla para medir" inferimos que un ingeniero en sistemas sería más bien una máquina (una máquina trivial).[3]

Aunque sabemos por Saussure que es "el punto de vista el que crea el objeto", en nuestras aulas se percibe lo contrario: un temor a que ciertos objetos nos hagan variar el punto de vista.[4] A producir lo que queremos evitar. Algunos otros términos como el de "cognición", también se consideran tabú. Los procesos psicológicos básicos, como la atención y la memoria, desdeñados como poco importantes, siguen siendo explotados por quienes sí los estudian en profundidad, para captar nuestra voluntad de consumidores. Y las metas del desarrollo tecnológico, continúan en manos de los mismos intereses, ya que nadie los disputa, porque "no es nuestro tema", porque el primer paso para refutar algo es haberlo comprendido. Y no puede comprenderse —como reza el epígrafe— aquello que no se ve con simpatía, o peor aún: que ni siquiera se ve.

---

1. En términos bachelardianos sería un obstáculo de cuarto tipo: unitario y pragmático. "El concepto de unidad es más pernicioso si va acompañado al de utilidad. Porque suele darse más valor y credibilidad a las explicaciones útiles. 'Nuestro espíritu -dice justamente Bergson- tiene una tendencia irresistible a considerar más claras las ideas que le son útiles más frecuentemente'" (Bachelard, 1938).
2. La introducción de técnicos en puestos políticos respondió a una búsqueda de supuesta neutralidad, saber objetivo o apoliticidad, a partir de la presidencia de Onganía: "La política dejaría el lugar a la administración con el resultante predominio de técnicos situados por encima de los intereses sectoriales y capaces de proponer e implementar las soluciones óptimas" (Cavarozzi, 2002).
3. Von Foerster denomina "máquina trivial" a una máquina predecible: siempre responde del mismo modo ante el mismo estímulo, mientras que las máquinas "no triviales", como el hombre, son creativas y pueden variar su respuesta aunque el estímulo sea el mismo.
4. La cita completa sería: "Lejos de preceder el objeto al punto de vista, se diría que es el punto de vista el que crea el objeto" (Saussure, 1976).

Los trabajadores del *software* no son responsables de los negocios que los incluyen, pero podrían dar cuenta de en qué consisten si fueran aceptados en la comunidad científica como investigadores y se reconociera su labor intelectual. Quienes al final de su vida laboral pueden objetivar la experiencia pasada, tras haberse sumergido en las complejidades de las organizaciones para las cuales se construye el *software* a medida, o ejerciendo la docencia, o simplemente padeciendo el desempleo, han sido protagonistas de la constante reinvención de una rueda que en ocasiones los eleva y otras veces los aplasta; están mucho más cerca de aportar una crítica profunda a las herramientas, al ejercicio de la profesión y a sus consecuencias sociales que quienes llegan a la tecnología con fines utilitarios y sólo perciben las capas superficiales. Sin embargo, las instituciones educativas que contratan analistas docentes suelen condicionar su participación pedagógica al punto de que se transformen en implementadores pasivos de las peticiones de los docentes de otras áreas, no como constructores del saber realmente integrados, ni como diseñadores de soluciones educativas con tecnología.

Pero el sesgo también se produce en el sentido inverso. El sector tecnológico se presenta a sí mismo desde un plano superior, como vocero de novedades y director de cambios en la educación. Cuando el empleo de las TIC no evidencia mejoras en el aprendizaje, rápidamente se atribuye la causa a los docentes. Según el proyecto de ciencias de la computación propuesto para la nueva escuela secundaria (NES), "no hay maestros con conocimientos adecuados", las clases de computación son dadas por no-especialistas", "la computación es percibida por los alumnos como algo de 'baja calificación'" y "poca gente elige carreras relacionadas con la computación" (Fundación Sadosky, 2013: 11).

La causalidad sería menos lineal si se la presentara como un problema complejo en el que intervienen múltiples factores. En primer lugar, deberían distinguirse las alternativas educativas que conducen meramente al consumo de medios, las que forman usuarios, las que promueven la programación como objetivo y las que apuntan realmente a la ingeniería (la programación como medio). Distinguirlas implica comprender los procesos cognitivos involucrados en el aprendizaje con medios digitales y trabajar en equipos verdaderamente interdisciplinares, en los que todas las disciplinas tengan el mismo peso en la participación, para llevar adelante proyectos de investigación educativa, no sólo de implementación de tecnología.

Se han conducido importantes líneas de investigación en psicología cognitiva en relación a los medios digitales y la educación, que se sitúan, grosso modo, entre dos posturas extremas y opuestas: por un lado, la sobreestimación del poder de los medios del enfoque nativos digitales (Prensky, 2001 y 2003), que contribuyó a naturalizar las consecuencias del consumo de medios a edades tempranas, y, por otro lado, la oposición tajante al uso de medios por considerarlos perjudiciales (Armstrong y Casement, 2000; Bowers, 2000; Torr, 2003). Entre ambas, algunos estudios cognitivos han formulado indagaciones más específicas para determinar qué medios, a qué edad, cuánto tiempo, qué estrategias pedagógicas favorecen el aprendizaje y cuáles lo obstaculizan.

Buckingham (2008) presenta una revisión de estos estudios, se opone al enfoque Prensky y propone un conjunto de prácticas educativas que denomina educación para los medios. Las investigaciones sobre exposición a los medios durante la primera infancia muestran no sólo que no contribuyen a mejorar el aprendizaje, sino que interfieren con el desarrollo (Anderson, Kirkorian y Wartella, 2008; Anderson y Hanson, 2010; Christakis, Ebel, Rivara y Zimmerman, 2004; Christakis y Zimmerman, 2006; Pediatrics, 2010; Powers, 2013; Rideout, 2013; Rideout, Foehr y Roberts, 2010; Roberts, Foehr y Rideout, 2005). Incluso una breve exposición al video ha mostrado tener un efecto inmediato en la interacción de los niños con sus juguetes (Kostyrka-Allchorne, Cooper y Simpson, 2017). Dejando a un lado el problema del acortamiento de los *spans* atencionales —una discapacidad adquirida—, si consideráramos sólo sujetos con buen rendimiento escolar, gran parte de la investigación sobre el aprendizaje de la programación en primaria y secundaria se centra en nuevos modos de enseñar programación con accesorios visuales y juegos estimulantes, pero el entusiasmo por el juego y el feedback positivo de los alumnos en estos cursos no debe confundirse con la evidencia de que realmente hayan aprendido algo distinto a lo que aprenderían en una clase tradicional (Buckingham, 2008; Costa y Miranda, 2016; Jenkins, 2001 y 2002; Jenkins y Davy, 2002). El uso de gráficos para mejorar la comprensión no es una cuestión de simple aplicación práctica, sino de complejas interacciones entre las estructuras superficiales perceptuales y las estructuras profundas semánticas —similares a las de comprensión de textos (Schnotz y Kulhavy, 1994; Schnotz y Baadte, 2014).

Las investigaciones en la línea de la teoría cognitiva del aprendizaje multimedia (Mayer, 2005) basadas en principios de aprendizaje —como el de canales duales para procesamiento visual o verbal, o el de capacidad limitada para el procesamiento— sugieren que los diseños instruccionales verdaderamente efectivos son los que se construyen a la luz de teorías psicológicas sobre cómo funciona la mente.

Un ejemplo de estas teorías aplicadas a la didáctica de la programación es el estudio de la transferencia de conocimiento en el pensamiento analógico. Ya en 1988 Perkins y Salomon advertían que para favorecer la transferencia en el aprendizaje, en especial en actividades de programación, se debían promover asociaciones conceptuales abstractas en el aprendizaje inicial que permitieran conectarlo con nuevos dominios. Sin estas estrategias de andamiaje, lo aprendido pasa a ser conocimiento inerte. Los conceptos básicos de programación aprendidos con lenguajes de programación visuales no se transfieren directamente a los lenguajes de programación basados en texto. La suposición de que aprender a programar podría tener un impacto positivo en la capacidad de razonamiento de los alumnos depende del tipo de estrategia pedagógica empleada.

Científicos cognitivos en nuestro país han propuesto estrategias para la recuperación analógica entre dominios que constituyen un importante aporte al problema de la transferencia de conocimiento (Trench y Minervino, 2017). Todo parece indicar que la relación entre mayor uso de las TIC y mejoras en el rendimiento no es directa. En 2015, la OECD realizó el primer estudio comparativo entre el uso de las TIC y el rendimiento de los estudiantes, en lectura digital, matemática y

comprensión de consignas en la web. Los recursos invertidos en TIC para la educación no están vinculados a un mejor rendimiento del alumnado en lectura, matemáticas o ciencia. En países donde no es común que los estudiantes usen internet en la escuela, el rendimiento en la lectura mejoró más rápidamente que en países donde el uso de Internet es más común (OECD, 2015: 146). Singapur y Corea, los dos países de mayor rendimiento en lectura digital y con un buen desempeño en la navegación web, tienen familiaridad con las computadoras; sin embargo, no están más expuestos a Internet en la escuela que el promedio. Esto sugiere que muchas habilidades de navegación se pueden adquirir más fácilmente si los estudiantes ya son competentes en los procesos de pensamiento de nivel superior y en el razonamiento en otros dominios (OECD, 2015: 187). En otras palabras, la capacidad para ser exitoso online podría deberse a estrategias que se adquieren offline o a través de una combinación de ambas.

Estos estudios aún no han sido considerados, probablemente por ser recientes, en propuestas de inclusión de tecnología como la de enseñar a programar en las escuelas, parte de la transformación curricular en la NES. Sin embargo, considerando la disparidad de objetivos entre las ciencias del cómputo y las ciencias de la educación, insistimos en la necesidad de ampliar nuestro conocimiento de los procesos cognitivos subyacentes a la programación y definir indicadores más precisos de su aprendizaje, así como del uso de las TIC, insertos en marcos teóricos más sólidos.

2. Competencia

El verdadero "motor" del desarrollo tecnológico es la competencia, un pretexto inagotable. En 1998, en el contexto del caso Estados Unidos contra Microsoft, la empresa afirmó que obligar a los usuarios de Windows a llevar en un mismo paquete también Internet Explorer es tan sólo una estrategia de "innovación" y "competición":

> "Si no hubiéramos desarrollado Windows, habríamos perdido el mercado del sistema a manos de un competidor como Apple […] Si no hacemos grandes progresos, los usuarios tendrán escasos incentivos para actualizar sus equipos, o comprarán productos de compañías rivales" (*Clarín Informática*, 4/2/98, cito en Laufer, 1998).

En los años 90 se vislumbraban nuevas formas de competir a través de la información:

> "... debido a que los cambios tecnología de la información se están volviendo tan rápidos e implacables y las consecuencias de quedarse atrás tan irreversibles, las compañías deberán modernizar y re-modernizar la tecnología o morir [...] tendrán que correr cada vez más de prisa tan sólo para quedar en el mismo sitio

> [...] El próximo escalón y la próxima arena para la diferenciación competitiva, gira alrededor de la intensificación del análisis [...] en un mundo competitivo donde las compañías tienen acceso a los mismos datos, ¿quién destacará en convertir datos en información [...]? [...] las decisiones que tomábamos en un mes, las haremos en una semana. Aquellas que tomábamos en una semana, las tomaremos en un día. Y las que tomábamos a diario las tomaremos cada hora" (Hopper, 1990).[5]

En la actualidad, las sugerencias para el futuro organizacional continúan en la misma línea: que el tiempo se utilice sólo para tomar decisiones, apegarse a las decisiones tomadas sin dar lugar a más debate, dejar de planificar para tomar decisiones directamente (Mankins, 2004; Mankins y Steele, 2006). Esta aceleración constante de la obsolescencia programada, donde lo único sustentable es la venta, promueve una carrera del *software* que poco tiene que ver con lo académico. Para participar en ella, ya lo anticipaba Dijkstra, no hace falta fabricar mejores productos, en tanto se pueda "engañar a la gente para que lo compre" ya que siempre es posible derivar nuevas versiones de lo mismo, "incluso con decimales -versión 2.6 o 2.7", cuando "la versión 1 pudo haber sido el producto terminado" (Dijkstra, 2000).

La tecnología hizo posible la modernidad, el estado de "sobremodernidad" (Augé, 2007; Bauman, 2000; Benasayag y Smith, 2010; Sibilia, 2005) y las formas de capitalismo actuales. Sin embargo, las prácticas futuras aún no han sido determinadas. No deberíamos promover un concepto de tecnología como entidad todopoderosa determinista, sino como la construcción de hombres que toman decisiones continuamente; por tanto, una construcción dinámica, discutible y modificable.

La apropiación de las fuerzas productivas se consigue mediante la apropiación de los instrumentos materiales —e intelectuales— de producción. El *software* es un producto ideal que, aunque se implementa físicamente en un *hardware*, no depende de las condiciones materiales del *hardware* para su diseño ni para su enseñanza. La potencia de su producción reside en la capacidad intelectual para la resolución de problemas que incluye, como primer paso, aprender a analizarlos. El trabajador del *software* debe prepararse para realizar tareas diversas, desafiantes, inesperadas, no sólo para la tarea tradicional de programación.

3. La escasez de ingenieros

La UNESCO Engineering Initiative (2011), junto a los Estados miembros, socios internacionales y expertos, trabaja en programas para fortalecer la educación en ingeniería. Según el informe sobre ingeniería (2010), la tendencia hacia un aprendizaje centrado en el alumno condujo a distintos enfoques, como el aprendizaje

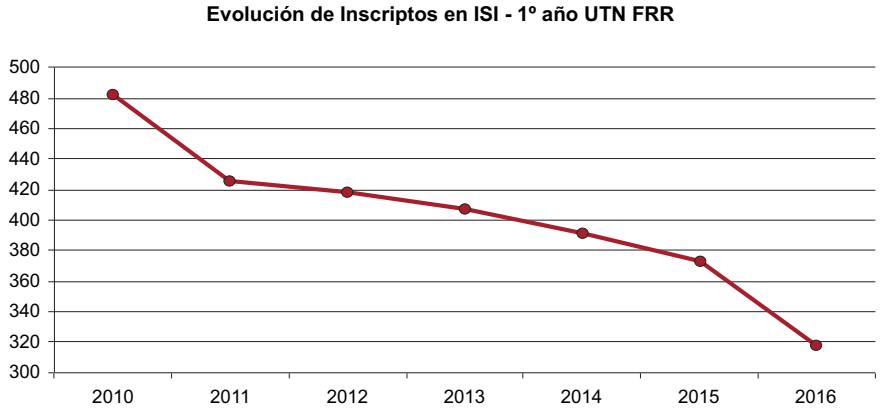
5. La traducción es del autor.

basado en problemas (PBL), con planes de estudio diseñados en torno a escenarios de problema considerados centrales. Según (Kolmos, Dahms y Du, 2010), pese a las propuestas de cambio, no parece haber verdadera transformación en las aulas. La Junta Europea de Acreditación para la Educación en Ingeniería (EUR-ACE) y la Junta Americana de Ingeniería y Tecnología (ABET) establecen nuevos criterios para la acreditación de la educación en ingeniería. Ambas instituciones de acreditación han formulado requerimientos de habilidades que van mucho más allá del conocimiento técnico. Según ABET:

- habilidad para interactuar en equipos multidisciplinarios
- habilidad para identificar y resolver problemas de ciencia aplicada
- comprensión de la responsabilidad ética y profesional
- habilidad para comunicarse efectivamente
- una educación amplia, necesaria para comprender el impacto de las soluciones en un contexto global y social
- el reconocimiento de la necesidad y la habilidad para participar en aprendizajes a lo largo de la vida
- el conocimiento de las problemáticas contemporáneas
- la capacidad para usar técnicas, capacidades y herramientas científicas modernas en las práctica profesional (Kolmos, Dahms y Du, 2010; Haase, Chen, Sheppard, Kolmos y Mejlgaard, 2013)

EUR-ACE define una variación de estas mismas habilidades poniendo mucho más énfasis en el aspecto intercultural (Kolmos, 2006). El saber de la ingeniería que siempre había consistido en un conocimiento basado en sistemas, donde los ingenieros trataban los problemas como cajas negras se va transformando lentamente en el nuevo desafío de trabajar en equipos interculturales e interdisciplinarios.

Figura 1. Número total de ingresantes por año a la carrera de ingeniería en sistemas de información



Fuente: Universidad Tecnológica Nacional, Facultad Regional Rosario

Según el Plan Estratégico de Formación de Ingenieros (PEFI 2012/2016) lanzado en Argentina en noviembre de 2012, impulsado por la Secretaría de Políticas Universitarias, se podía llegar al objetivo general de incrementar la cantidad de graduados en ingeniería generando vocaciones tempranas, incrementando la retención en el ciclo básico, y mejorando los indicadores académicos, entre otros objetivos planteados, no cumplidos según los plazos proyectados.

Consideramos que, si bien los incentivos económicos son importantes, hay factores de deserción y de disminución del interés por la ingeniería que están ligados, por un lado, a sesgos comunes en educación, producto de diferencias psicológicas, culturales y epistemológicas: la cultura universitaria de las ciencias formales promueve creencias acerca de las capacidades naturales para dichas ciencias. Sin embargo, el cálculo infinitesimal, la cibernética e incluso las matemáticas modernas son innovaciones culturales recientes desde un punto de vista evolutivo; su dominio no depende de habilidades heredadas genéticamente, ni se desarrolla en contextos de crianza, sino en contextos de enseñanza formal, ya que se trata de procesos conscientes y voluntarios (Rivière, 1999; Vygotski, 1978). Por otro lado, a las campañas incansables de los empresarios visionarios para captar el mercado educativo como trampolín al mercado hogareño, mucho más lucrativo (Buckingham, 2008), han logrado confundir tecnología con medios y puesto en tela de juicio el sentido mismo de las instituciones educativas y de los saberes que imparten.

Aceptamos que deben producirse cambios en el modo de enseñanza a favor de aprendizajes centrados en el alumno en entornos que podrían incluir tecnología, pero no es ella misma la que produce la transformación. Las decisiones en educación han sido históricamente tomadas para conveniencia de los estados y de los hombres en tanto ciudadanos, antes que para su crecimiento personal como sujetos libres. Ahora

además, se imbrican otros intereses en una trama que dificulta su identificación. Si el cambio beneficiará o no a la educación en tanto formadora de sujetos, dependerá de que se les brinde a los alumnos la oportunidad de elegir y a los futuros docentes una formación integral (no sólo técnica), cuyo centro de interés sean los hombres que programan, no sólo los programas.

Comprender el álgebra del Boole permite escribir programas que tengan sentido para el ordenador. Pero no poder explicarnos nosotros mismos la cadena de sentidos que va desde los artefactos al hombre, y describir con coherencia la propia participación en el proceso, es quedar diluido en un sentido ajeno.

Se promueve un supuesto cambio conceptual en la relación entre los hombres y las máquinas. Pero, al ser las máquinas producidas por empresas, se trata de una relación hombre-empresa entre hombres que poseen medios de producción tecnológica y otros que deben adquirirlos como materia prima para su trabajo. Por tanto, deben distinguirse los distintos niveles posibles de apropiación de herramientas de *software* según el grado de dependencia que cada productor tiene con su proveedor. Si el cambio curricular sólo atenderá a las habilidades digitales promovidas por el uso descontextualizado de *software online*, esto tendría como consecuencia positiva para la empresa —incrementar el número de usuarios autodidactas y consumidores— agregando esta vez los de un nuevo tipo: los nativos programadores. Pero cabe la pregunta de si ese tipo de programadores estará motivado para iniciar carreras de ingeniería.

Cuanto más cercano es el vínculo con la máquina, más difícil es percibir la metáfora en la que se está inmerso: la computadora como una mente sólo ha constituido un problema para los psicólogos cognitivos de la inteligencia artificial. En la vida cotidiana de las personas es más palpable la analogía entre la computadora y un esclavo: la estructura de control que le da sentido al cómputo es la iteración. Repetición de la misma operación millones de veces si es necesario, sin manifestación de agotamiento, sin errores admisibles, por un costo cercano a cero y pudiendo ser reemplazada en cualquier momento sin afectar el sistema.

Más claras que las metáforas son las paradojas: en los países consumidores como Argentina, la proliferación de artefactos tecnológicos va en aumento al tiempo que disminuye el interés por las carreras ingenieriles, base de la producción de dichos artefactos.

4. ¿La escasez de programadores es real?

Si bien en apariencia es innegable que el mundo empresarial se mueve a la velocidad de la luz, esto sólo es cierto en algunos aspectos: las tecnologías evolucionan rápidamente, las reuniones del personal se suceden una tras otra, la comunicación es instantánea; sin embargo, a medida que las personas transitan la cotidianeidad de una organización, la productividad decae, las estadísticas muestran un cese del crecimiento desde 2007. El tiempo y el esfuerzo para completar muchas tareas críticas de negocios aumentó significativamente entre 2010 y 2015. La sugerencia de

los líderes: contratar a los principiantes ágiles, nativos digitales del cambio vertiginoso que prometen desplazar la lentitud de aquellos que, por el contrario, generan el "arrastre organizacional" que amenaza con destruir la productividad; es preciso aprovechar los "secretos" de la productividad del capital humano: que nadie pierda tiempo, que se activen todo tipo de herramientas y procedimientos que alienten la acción rápida y, al estilo Netflix, atraer a los mejores talentos, poner a esas personas "especiales" en el sector más productivo para que inspiren al resto (Mankins, 2004; Mankins y Garton, 2017).

La escasez de programadores es un título equívoco para expresar una preocupación más precisa: la escasez de programadores muy jóvenes, muy talentosos y formados por instituciones externas a las empresas que los solicitan. ¿Qué pasará con el resto? Probablemente serán los impulsores no asalariados más importantes de la demanda: usuarios altamente calificados para trabajar como analistas de sus propios productos (véase la quinta sección) o consumidores que generan sus propios contenidos.

Tradicionalmente, las empresas contrataban estudiantes, se hacían cargo de su formación inicial y, con el transcurso del tiempo, también de su actualización. Trabajando en relación de dependencia, cada cambio tecnológico implicaba reeducar al personal adulto, ya habituado a sus tareas. Resulta mucho menos costoso contratar continuamente jóvenes recién egresados o estudiantes que, altamente motivados con expectativas no realistas pero convenientes, aceptan sueldos que nunca son buenos para los que recién se inician (ni en empresas como Google) ni tienen cargas familiares ni la necesidad de un ingreso fijo. Las empresas fueron optando por tercerizar este servicio a agencias que se ocupan del reclutamiento de personal. Algunas empresas líderes ni siquiera contratan programadores; contratan el servicio de programación para dedicarse de lleno al análisis, diseño e ingeniería del *software*.

Debería hablarse, en realidad, de una escasez de talentos renovables, cuya búsqueda es la principal forma de compensación por la pérdida de productividad empresarial. Y este es el principal factor de competitividad. El personal debe ser sumamente ágil y con un alto grado de motivación, aunque ella decaiga en el corto plazo y haya que pensar en reemplazarlos.

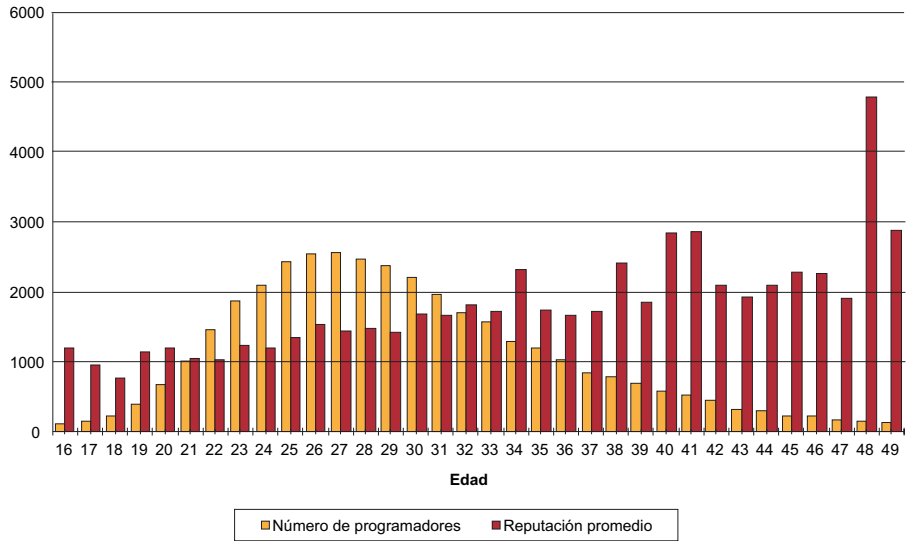
Si estos talentos ágiles surgen entre la multitud, hay que capacitar a todos para elegir sólo a algunos. Semejante costo no lo puede asumir el sector empresarial.

Joel Spolsky, CEO de StackOverflow y fundador de Fog Creek Software, habló en una conferencia en Talent Leaders Connect (TLCon) 2014 —el programa de eventos más grande del Reino Unido para profesionales de recursos humanos dedicados a captar talentos— acerca de las mejores estrategias para lograr que los mejores programadores acepten trabajar en sus empresas.[6] Aconseja captarlos generando un

6. Más información disponible en: https://es.stackoverflow.com/, https://www.joelonsoftware.com y https://www.youtube.com/watch?v=yLyALWAp8IM.

ambiente de trabajo adecuado a la personalidad de los programadores, que es bastante opuesta a la personalidad de los CEO, ya que se trata, según él, de personas "poco sociables", que prefieren tratar con computadoras porque son elementos "predecibles" —a diferencia de los seres humanos— y que no desean recibir llamados telefónicos, entre otros rasgos característicos de su perfil. Su audiencia son profesionales que, como él, se dedican al reclutamiento de personal en empresas de *software*, pero es importante destacar que no buscan cualquier programador sino a los mejores talentos.

Figura 2. Número de programadores y su reputación, por edad

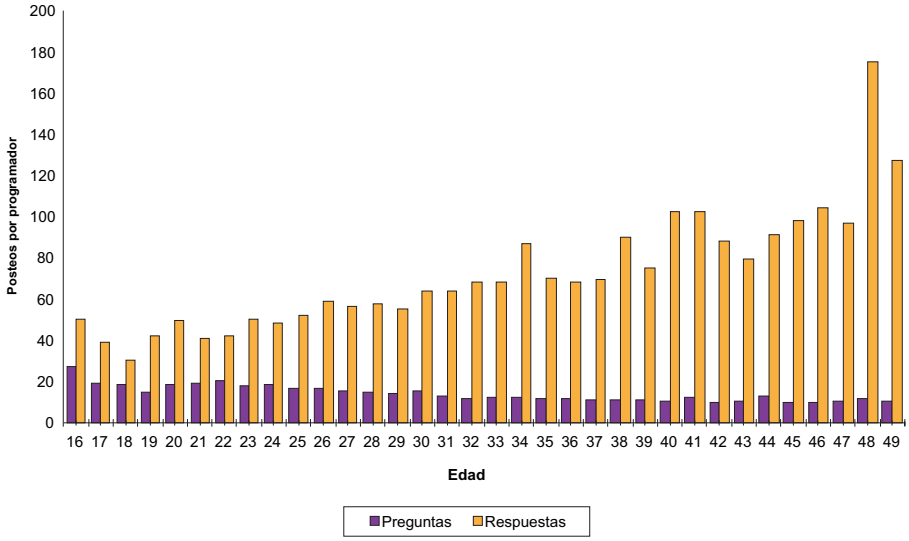


Fuente: Stackoverflow

En su blog y en una conferencia en Yale cita los datos obtenidos por Spolsky —también accesibles en el blog de Peter Knego sobre desarrolladores—, Robert Martin discute acerca de la relación entre la edad y la reputación de los programadores.[7 8 9] Es evidente que a mayor edad hay un mejor desempeño del programador. Esto se mide con un cuestionario de preguntas técnicas. Los mayores responden muchas más preguntas que los programadores jóvenes.

7. Más información disponbible en: http://blog.cleancoder.com/uncle-bob/2014/06/20/MyLawn.html y https://www.youtube.com/watch?v=TMuno5RZNeE.
8. Más información disponible en: https://docs.google.com/spreadsheets/d/1Xc2WDW6-D3EB-nyqqbkXI0xcF_rVZfjn3W5WKP7jJIA/edit?hl=en_GB&pli=1&pli=1&hl=en_GB&hl=en_GB#gid=2.
9. Más información disponible en: http://coding-and-more.blogspot.com.ar/2011/06/its-official-developers-get-better-with.html.

Figura 3. Preguntas y respuestas, por edad



Fuente: Stackoverflow

Es evidente también que los programadores jóvenes tienen un estilo más impulsivo, menos reflexivo que el de los mayores (Martin, 2014). Sin embargo, la industria prefiere contratar a los jóvenes. El análisis de Martin no apunta a identificar la gente más talentosa, sino al número de programadores en general y sus edades. Al modo en que la industria del *software*, en pos de ganar productividad, se va desprendiendo de la invaluable sabiduría de los adultos que, como todo lo "viejo", parece no tener lugar en la tecnología. La mayoría de los programadores tienen menos de 28 años. La mayoría de los programadores tienen menos de seis años de experiencia. En 2014 había aproximadamente 20 millones de programadores (Avram, 2014); en 1974 eran menos de 100.000 (Martin, 2014), lo que significa una tasa de crecimiento del 14,5%. Cada cinco años se duplica el número de programadores. Las consecuencias para la industria son la falta de líderes y la industria del *software* en una inmadurez perpetua mientras la curva siga en crecimiento. Además del hecho de tener que reaprender incesantemente lo mismo que ya se había aprendido cinco años antes (Martin, 2014).

Las consecuencias para la educación: con una expectativa de futuro laboral tan breve, a sabiendas de que ningún programador conserva su puesto a largo plazo, aun atravesando el exigente filtro de ser "el mejor", la formación técnica debería apuntar a la ciencia o la ingeniería. No tiene sentido formar a los niños para ser específicamente programadores, ocupando el espacio de otras asignaturas con la promesa de que las habilidades metacognitivas se transferirán a otros dominios. Ya hemos analizado que la transferencia exitosa depende del grado de abstracción del aprendizaje. Esto podría lograrse, durante las actividades de programación, poniendo

el énfasis en la representación del problema, en la comprensión lectora de las consignas, en la vinculación con problemas abiertos, tanto de ciencias naturales como sociales —tan familiarizadas con la complejidad—, como un camino más seguro hacia las vocaciones en ingeniería. En otras palabras, hay que ayudarlos a discernir entre las dos formas de aprender a programar: programar como objetivo o programar como medio.

5. La desaparición del análisis

Entre las décadas del 70 y 80, durante el surgimiento de las primeras carreras universitarias de informática latinoamericanas (Aguirre y Carnota, 2009), los títulos de analista programador y de analista de sistemas ofrecidos por las tecnicaturas denotaban la valoración de las incumbencias: el análisis en primer lugar. Aprender a programar implicaba iniciarse en la comprensión de problemas moderadamente complejos. Algunos docentes universitarios le infundían a la programación un carácter científico, incluyendo como contenido del material de cátedra técnicas de resolución de problemas y un análisis exhaustivo de las soluciones (Mastrogiuseppe y Iwanow, 1992).

La teoría dominante del análisis estructurado de sistemas apuntaba un análisis riguroso de los requerimientos del *software*, su documentación, la verificación de los requerimientos con el usuario, el diseño, la implementación, las pruebas del sistema y las modificaciones. Se suponía que la rigurosidad en la documentación y en el análisis aumentaba las posibilidades de éxito futuro de las implementaciones de proyectos.

Con la masificación de los productos de *software*, la mejora continua y el aumento del tamaño de los programas, el tiempo invertido en modificaciones por depuración o por actualización de versiones superó ampliamente al tiempo invertido en el diseño de la solución (Pressman, 2010), hasta que dicha etapa de diseño quedó eliminada del proceso. Las metodologías ágiles como Scrum proponen un trabajo en equipo de desarrolladores con habilidad para analizar problemas, pero la parte esencial del análisis la realiza el mismo usuario que debe necesariamente estar comprometido con el desarrollo. Las modificaciones constantes se realizan a plazos semanales y gran parte de la depuración se efectúa de modo automático. En este contexto, las habilidades que se esperan del desarrollador apuntan a una comunicación fluida con usuarios, habilidades de razonamiento, trabajo en equipo y capacidad para adaptarse rápidamente a situaciones cambiantes, esto es: para transferir sus conocimientos de un entorno a otro.

La alta disponibilidad de aplicaciones web para aprender a programar, así como de lenguajes con posibilidad de intercambio en la red (C++, Python, Java), en las clases de programación iniciales, ha desplazado el foco de atención hacia la codificación en máquina, en desmedro de la etapa previa de representación del problema a resolver. El análisis, en tanto comprensión del problema, también tiende a desaparecer.

Otra cuestión que contribuye a dicho desplazamiento proviene de la forma en que las clases son dictadas por los profesionales que trabajan diariamente como

programadores. Un problema común en la enseñanza de habilidades procedurales es que el experto ya no puede acceder al estado declarativo de su propio conocimiento a menos que dedique el tiempo suficiente a la explicitación paso a paso de su propia práctica. Esto significa que el docente debería ser un profesional que ha sumado una nueva formación a la que ya tenía, de no ser así, el experto tiende a acelerar los tiempos de aprendizaje de sus alumnos salteando etapas.

A nivel teórico, la solución de problemas implica distinguir fases. La más importante es la de representación del problema (Carretero y Asensio, 2008; Holyoak y Morrison, 2005; Newell y Simon, 1970; Pérez Echeverría, 2008; Polya, 2014). Según Chi (1989), las diferencias en el rendimiento de los alumnos se ven en esta etapa fundamental (Chi, Bassok, Lewis, Reimann y Glaser, 1989; Renkl, 1997); sin embargo, la mayoría de las teorías sobre resolución de problemas se han focalizado en el proceso de compilación (Anderson, 1987), relegando a un segundo plano el paso previo de construcción de la representación. Es en esta instancia donde interviene el análisis de los requerimientos del sistema que serán programados y de la obtención de la solución algorítmica. Aquellos alumnos que invierten más tiempo en explicarse a sí mismos aquello que intentan comprender son más eficaces en la resolución de problemas y en la adquisición de nuevos conceptos (Aureliano, Tedesco y Caspersen, 2016; Chi, Bassok, Lewis, Reimann y Glaser, 1989; Kastens y Liben, 2007; Mitchell, Mertz y Ryant, 1994). En la solución de problemas algorítmicos en particular, si bien admitimos que las fases no son estrictamente secuenciales sino recursivas (Pennington y Grabowski, 1990), se tiende a confundir cada vez más la fase de comprensión del problema en su dominio con la del diseño del algoritmo.

Los alumnos novatos necesitan más tiempo de elaboración en la fase declarativa, durante la cual deberían probar modos de explicar los problemas en sus propios términos, utilizando sus saberes previos. En cambio, se los induce a una automatización prematura, a confundir "ejercicios" con "problemas" (Véase Pérez Echeverría y Pozo, 1994, cito en Carretero y Asensio, 2008).

6. ¿Qué propuesta elegir?

Un criterio de evaluación de las distintas propuestas educativas para la formación de los futuros trabajadores del *software* podría ser el nivel de profundidad conceptual que apuntan a desarrollar en los sujetos. Algunas parecen estar más dirigidas a la formación de programadores o usuarios, mientras otras sientan las bases para perfiles profesionales universitarios e investigadores.

Figura 4. Números binarios

Fuente: csunplugged.org

Figura 5. *Sorting Networks*

Fuente: csunplugged.org

Lo que genera confusión es que todas las propuestas se presentan como "la solución al problema de la escasez de profesionales del *software*". Y todas coinciden en abogar por una ciencia de la computación que no debe confundirse con el uso de las TIC. Sin embargo, es importante mencionar que algunas alternativas difieren en un punto esencial: la edad a la cual los niños deben iniciarse en la manipulación de

medios de pantalla. Mientras un sector sostiene que los niños deben operar, desde el jardín de infantes, con dispositivos de pantalla, el otro sector afirma, apelando a las bases de la teoría de la computación, que los contenidos fundamentales de la computación y la ingeniería no dependen ni mejoran con el uso de dispositivos. Más aún, que algunas de estas ideas son mejor transmitidas prescindiendo de ellos, que "la ciencia de la computación trata fundamentalmente sobre algoritmos" y lo que hace vívida la práctica no es mirar pantallas sino introducir los conceptos computacionales y matemáticos en las experiencias cotidianas, con narraciones y otras actividades. "El núcleo intelectual de la ciencia de la computación puede ser presentado a los niños incluso en situaciones donde no hay computadoras. […] Muchas de las ideas nucleares de la ciencia de la computación se introducen mejor sin computadoras" (Fellows, 1991). En este enfoque, que pretende desarrollar habilidades desde la infancia que despierten vocaciones por la ingeniería y la ciencia, se propone situar al alumno en contextos cotidianos, priorizar el significado a la información, explorar el sentido antes que la sintaxis de las reglas (Bell, Rosamond y Casey, 2012; Fellows, 1991; Fellows y Parberry, 1993; Rapaport, 2015).

Tabla 1. Dos maneras de aprender a programar

Programar como objetivo	**Programar como medio**
Nuevas TIC. *Software* para aprender a programar como estímulo	Enfoque universitario hacia las ciencias de la computación y la ingeniería
Aprendizaje procedimental	Aprendizaje conceptual. Distinción entre etapa declarativa (comprensión del problema) y procedural
Orientado a las recomendaciones de (Microsoft) y el International Computer Sciences Education Standards (ICSES) en concordancia con el Common Core	Orientado a las recomendaciones de CSUnplugged.org, patrocinado por Google y CS Education Research Group en la Universidad de Canterbury, Nueva Zelanda
Enfoque instruccional	Enfoque constructivista
Pensamiento computacional según Wing (2006 y 2009), basado fundamentalmente en abstracción, descomposición y deducción	Pensamiento computacional según Papert (1980), basado fundamentalmente en la producción de conocimiento nuevo (constructivista)
Problemas cerrados	Problemas abiertos
Propone que los niños se habitúen al uso de pantallas desde los tres años	No requiere el uso de computadoras

7. Cambios de paradigma: la necesidad de una psicología del aprendizaje

Es evidente que la mayoría de los profesionales del cómputo simpatizan con Piaget. Las tareas piagetianas pertenecen al ámbito de la ciencia experimental y lógico matemática. Estudios populares sobre enseñanza de la programación y del pensamiento computacional a niños se basan en la teoría piagetiana (Guzdial, 2004;

Papert, 1993) o bien en un marco epistémico cartesiano y de lógica formal (Wing, 2006). Las citas a Piaget apuntan a un constructivismo entendido como teoría psicogenética, no al constructivismo como teoría del conocimiento. Los autores coinciden con Piaget en que el aprendizaje es activo, esto es, construido por el mismo sujeto, pero subyace una noción de que la estructura lógica deductiva de los programas es también —o debería ser— la forma en que se estructura nuestro pensamiento.

Investigadores como Arblaster (1982) afirman que la semántica de las soluciones algorítmicas no es independiente del lenguaje mediante el cual se expresan. Es decir que los programas son, ante todo, soluciones expresadas mediante un lenguaje. Podríamos enriquecer esta visión agregando que en el acto de programar intervienen dos lenguajes: uno artificial y otro humano (el lenguaje del "resolvedor" del problema, que no sólo resuelve, sino que se explica a sí mismo cómo resuelve). Este aspecto de la programación no ha sido explorado en profundidad.

Sin embargo, como la mayoría de los investigadores asocian la algorítmica con la matemática, encuentran en Piaget la inspiración para idear estrategias sobre cómo los niños deberían aprender a programar, que afectan a la didáctica y al mismo concepto de pensamiento computacional que se pretende instaurar, ya que se omiten las objeciones y rectificaciones a la teoría piagetiana. Se sugiere a los alumnos "pensar de modo estructurado" o "estructurar su pensamiento" en el sentido de organizarlo (Wing, 2006 y 2009). Circula la idea de que pensamiento computacional es sinónimo de pensamiento formal.

Piaget e Inhelder (1955) sostuvieron que a partir de los 11 y 12 años se inicia el estadio de las operaciones formales. Estas operaciones tienen características funcionales y estructurales. Dos de las características funcionales son el razonamiento hipotético-deductivo y el razonamiento proposicional. A partir de 1970, la escuela de Ginebra rectificó la posición piagetiana admitiendo que no todos los sujetos llegan a las operaciones formales del mismo modo, que depende de sus aptitudes y sus especializaciones profesionales. En definitiva, se reconocía que el pensamiento de adolescentes y adultos no funciona solamente basándose en la estructura de los problemas, sino también en el contenido, que el pensamiento no sería solo "formal". El supuesto déficit en las operaciones formales podía explicarse por: la incidencia de la tarea, la incidencia del conocimiento previo, la inconsistencia del modelo de lógica pura, y la importancia de la metacognición y control epistémico (Carretero, 2008).

Quedó demostrado que las operaciones formales no son la cima del desarrollo intelectual. Parecen coexistir en el propio individuo otros modos de pensamiento, que se han agrupado con el nombre de "pensamiento posformal", que admiten la posibilidad de un conocimiento relativo que acepta la contradicción y un sistema más abierto que el de la física newtoniana de las tareas piagetianas (Carretero, 2008; Carretero y Rodriguez Moneo, 2008; Kramer, 1983). Además, las reglas del pensamiento proposicional no son formales, sino sensibles a los contenidos y al contexto. La oposición tradicional entre teorías sintácticas versus teorías semánticas dio lugar a un cambio en el énfasis de los estudios cognitivos pasando de la sintaxis

a la semántica, acentuando la importancia del contenido de las representaciones por sobre las reglas lógicas (Cheng y Holyoak, 1985; 1989; Cheng, Holyoak, Nisbett y Oliver, 1986; Cosmides, 1989; Johnson-Laird, 1975).

Estos avances en el conocimiento del razonamiento humano no se vieron reflejados en las ciencias formales ni en las ciencias de la computación en particular. Abundan ejemplos sobre cómo los cambios de paradigma computacionales impactaron negativamente en el aprendizaje por la persistencia de un modelo de razonamiento humano formal y deductivo. El ejemplo que se describe a continuación es una experiencia sobre el pasaje de la programacion lineal a la estructurada en los 80.

Entre fines de los 80 y principios de los 90, las cátedras de algorítmica básica anunciaban e implementaban un cambio de paradigma (que no sería el último) en la programación y el análisis de sistemas.[10] Los algoritmos debían ser más cortos, más claros y modulares, ya no una secuencia interminable de comandos concatenados en forma caprichosa, sino una breve lista de llamados a procedimientos y funciones también concisos, que mantuvieran cierta coherencia y cohesión.[11] Se actualizaba la currícula para los contenidos de dichas asignaturas, pasando de la programación lineal —o monolítica— a la estructurada.[12] En algunos casos se brindó a los estudiantes la posibilidad de cursar la misma materia por segunda vez introduciendo dicho contenido en cátedras posteriores, pero cambiando radicalmente su modo de trabajo. Los que ya tenían experiencia como programadores, por el primer año de carrera transcurrido o porque trabajaban en el área, se vieron obligados a repensar sus programas, pero sobre todo a "pensarse a sí mismos" programando.

Aunque existían métodos inductivos como el llamado *bottom-up*, en general fueron contraindicados y las cátedras adoptaron la metodología deductiva o *top-down* (Dijkstra y Dahl, 1972; Wirth, 1976), que consistía *grosso modo* en la aplicación del método cartesiano: dividir la solución *a priori* en bloques de información generales, a continuación dividir cada uno de estos bloques en procedimientos más elementales, y así sucesivamente hasta el mínimo nivel de detalle, que sólo se apreciaba al final del procedimiento, ya que estaba contraindicado prestar atención a los detalles en el momento inicial de la fragmentación. Supuestamente, atender al árbol nos impediría ver el bosque.

El problema del programador consistía en lograr comprender una especificación en lenguaje humano, resolver el problema y codificar la solución en un lenguaje artificial. Con lenguaje humano nos referimos al habla, transcripciones de entrevistas a

10. Tradicionalmente "programación lógica", "algoritmos" y "estructuras de datos", "programación I" o denominaciones similares. Se diferencia de la programación en lenguajes específicos en que sienta las bases lógicas para redactar cualquier programa procedural.
11. Lo de "caprichosa" refiere, sobre todo, al uso indiscriminado de la sentencia GOTO, a lo cual se oponían rotundamente tanto Wirth como Dijstra, aunque Knuth admitía su utilización (Knuth, 1974). GOTO permite efectuar saltos entre puntos arbitrarios del programa, desde cualquier acción que se esté ejecutando a cualquier otra. Es una libertad en el momento de la programación con un alto costo de mantenimiento.
12. La oposición lineal-estructurado no implica que el paradigma estructurado fuera no lineal. Ambos métodos eran secuenciales.

usuarios o especificaciones hechas por analistas, mientras que el lenguaje artificial es un lenguaje procedimental de rigurosa precisión que no admite expresiones coloquiales ni ambigüedades.[13] La diferencia no es sólo formal, sino de contenido: en lenguaje humano se expresa la vida de una organización, que incluye las vidas de sus miembros —futuros usuarios del sistema— cuyo relato abarca desde el quehacer cotidiano a las anécdotas (cuando no los enmudece el temor a que el nuevo sistema los reemplace). Y el relato en lenguaje artificial —el texto del programa— está dirigido a un ordenador. Cuanto más estructurado, claro y legible sea, mayor es la posibilidad de objetivación y mantenimiento.

Pero la supuesta capacidad para la abstracción se transformó en el principio explicativo, similar a la fuerza de gravedad, que todo lo explica pero no explica nada.[14] Un acuerdo entre docentes y alumnos para establecer dónde dejar de preguntar. Algunos alumnos sencillamente tenían un "instinto" de programadores. Pero lo que más nos intrigaba era por qué, en este salto desde la libertad a la deducción estricta, ciertos alumnos excelentes programadores en otro paradigma fracasaban rotundamente en el estructurado. Algunos de ellos, muy observadores y analíticos, estaban convencidos de que la "capacidad" obraría por sí misma, pero las "ideas claras y distintas" no aparecían. Las musas inspiradoras no hablaban en PASCAL.

Gran parte del alumnado solía reprobar esta asignatura, caso típico de materia "filtro", tautología cotidiana entre ingresantes (la razón por la que una materia se hace difícil de aprobar es: "Porque la materia es 'filtro'"). Lo que llamaba nuestra atención es que algunos estudiantes obtenían mejores calificaciones precisamente pasando por alto algunas reglas del método, en pos de una solución más intuitiva. Por cuestiones vinculadas a sus saberes, dedicaban más tiempo a la fase que Polya (2014) define como "comprensión del problema". Por un lado, escribían para comprender, describiendo casos a manera de simulación de situaciones desconocidas propias de un determinado dominio, definiendo o explicando lo que no comprendían. Alcanzaban la abstracción de un modo inductivo, dando un paso intermedio no aceptable en una cultura computacional asociada con la rigurosidad, la formalidad y la síntesis.

La crisis del *software* fue una crisis en el costo de desarrollo —el trabajo humano detrás de la máquina— excesivamente caro en comparación con la fabricación de los componentes electrónicos. Optimizar la forma de trabajo (el estilo de programación) devendría en una baja de los costos. Estandarizar los códigos aumentaría la comunicabilidad entre equipos numerosos de personas. Paralelamente, la estandarización también devino en un cambio en los requerimientos para ser programador y, por tanto, en una reducción del número de las personas que se insertaban exitosamente en dicha disciplina. Podría decirse que se acortaron ciertas libertades humanas, pasando a estrategias apriorísticas de programación. Pero

13. Ejemplos: una simple charla con los futuros usuarios del sistema, o bien un enunciado escrito planteado por el profesor en forma deliberadamente desordenada.
14. El autor refiere al sentido de principio explicativo que da Bateson (1972).

quienes hayan sido testigos o participado en conversaciones con programadores "lineales" habrán notado en ellos una actitud casi lúdica hacia su trabajo, al que consideraban una producción personal. A pesar de que lidiaban con el desorden, el programa les pertenecía.

Lo que algunas versiones del pensamiento computacional denominan "abstracción", haciendo referencia a una capacidad formal de propósito general (Wing, 2006; Wing, 2009), en otras versiones se piensa como una capacidad dependiente del dominio y la experticia. Los estudiantes necesitan experiencias concretas en un entorno antes de poder comprender abstracciones propias de ese dominio (Guzdial, 2004). De hecho, el tipo de abstracciones que se utiliza en el paradigma procedural (procedimentales) difiere de las abstracciones en el paradigma de objetos (conceptuales).

Con la evolución del modelo de objetos, las abstracciones claves pasaron a ser los datos en lugar de los procesos. La tendencia fue alejarse de los lenguajes que le dicen a la computadora qué hacer (imperativos) hacia los que describen las abstracciones clave en el dominio del problema (declarativos) (Booch, 1994). En los nuevos paradigmas, alcanzaría con saber cuál era la demanda y a qué objeto iba dirigida —algo que nos resulta cada vez más familiar en lo cotidiano. De ahora en más, se premiarían las habilidades de selección y descarte.

La necesidad de una psicología del aprendizaje que indague los procesos cognitivos subyacentes a la programación de ordenadores es evidente. La psicología de la programación, si bien no constituye una disciplina, surge en los años 70 como campo de interés (ESP - Empirical Studies of Programmers, 2002/2017; Hoc, Green, Samurçay y Gilmore, 1990; Psychology of Programming Interest Group, 1987/2017; Sajaniemi, 2008; Soloway y Spohrer, 1989; Weinberg, 1971) a partir de enfoques normativos desarrollados por informáticos. Recién en la obra de Weinberg (1971) se postula que, en cualquier enfoque de la programación, debería incorporarse un punto de vista psicológico.

Lo que intuitivamente habían resuelto los alumnos en aquellas clases, tomando notas, se estaba confirmando parcialmente en experimentos realizados por psicólogos cognitivos que iniciaban investigaciones en el campo informático. Turkle y Papert (1990) comparaban estilos de aprendizaje de la programación, distinguiendo la tendencia *bottom-up* —desde las partes al todo— en alumnos con estilos *bricoleur*, de la tendencia *top-down* en otros alumnos. Knuth (1993) diseñaba un *software* para edición de programas que ofrecía la posibilidad de elegir la alternativa que es "psicológicamente mejor" para cada programador, sea *top-down* o *bottom-up*, con la convicción de que el diseño computacional no equivale al modo en que los programadores piensan. El pensamiento no funciona como un ordenador. Y aunque la programación estructurada permite diseñar programas ordenados y claros, para llegar a esa solución final es posible que algunos sujetos tomen atajos o desvíos distintos (Knuth, 1993).

Se admitió que la programación estructurada ayuda a reflejar la complejidad de la relación entre módulos globales y particulares, pero no refleja el modo en que los programadores piensan los programas. Estudios cognitivos en ambientes de desarrollo *top-down* mostraron que incluso los programadores expertos no son capaces de ajustarse a este tipo de estrategia; más bien utilizan estrategias "oportunistas" que combinan deducciones *top-down* con inducciones *bottom-up*, lo que condujo a diseños más eficientes de los lenguajes (Hoc, Green, Samurçay y Gilmore, 1990).

8. Conclusiones

La demanda de un mayor número de programadores disponibles, por la brevedad de su rendimiento laboral óptimo, es un requerimiento empresarial que se desprende de la agilización de los métodos en TI que implicaron una mejora para dicho sector, pero que no se traducen directamente en mejores oportunidades educativas o laborales.

A la educación le compete la demanda de futuros científicos, ingenieros e investigadores para un desarrollo sustentable. Por lo tanto, no es necesario transferir la aceleración en TI a la educación, saltando etapas de comprensión y profundización del conocimiento, o reduciendo la edad mínima requerida para el contacto de los niños con los medios digitales, ya que los tiempos requeridos para el desarrollo del razonamiento humano siguen siendo los mismos. Aunque se producen avances tecnológicos en *hardware* y *software*, las ciencias de la computación como disciplina, y su didáctica, aún se encuentran en sus inicios. Se necesita la participación de la filosofía para delimitar su objeto de estudio, de la psicología para hallar las mejores estrategias de enseñanza sin afectar el desarrollo humano, y de las ciencias sociales para comprender su historia y las repercusiones del diseño y uso de las inteligencias artificiales. Caso contrario, sin la colaboración interdisciplinar y humana, el rumbo lo dirigirá el mercado.

Bibliografía

ABELSON, H. (2005): "6.001 Notes: Section 1.1. Introduction to computation", en S. E. Grimson (ed.): *6.001 Structure and Interpretation of Computer Programs,* Massachusetts Institute of Technology, MIT OpenCourseWare. Disponible en: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-001-structure-and-interpretation-of-computer-programs-spring-2005/lecture-notes/lecture1webhand.pdf.

ABELSON, H., SUSSMAN, G. J. y SUSSMAN, J. (1996): "Preface to the First Edition", en H. Abelson, G. J. Sussman y J. Sussman (eds.): *Structure and Interpretation of Computer Programs,* MIT Press, pp. 11-13. Disponible en: https://mitpress.mit.edu/sites/default/files/6515.pdf.

AGUIRRE, J. y CARNOTA, R. (2009): *Testimonios, Historia de la Informática en Latinoamérica y el Caribe: Investigaciones,* Río Cuarto, Universidad Nacional de Río Cuarto.

AMERICAN ACADEMY OF PEDIATRICS (2010): *Policy Statement—Media Education,* vol. 126. DOI: 10.1542/peds.2010-1636.

ANDERSON, D. L., KIRKORIAN, H. L. y WARTELLA, E. A. (2008): "Media and Young Children's Learning", *Journal The Future of Children,* vol. 18, n° 1. Disponible en: http://futureofchildren.org/futureofchildren/publications/docs/18_01_FullJournal.pdf.

ANDERSON, D. R. y HANSON, K. G. (2010): "From blooming, buzzing confusion to media literacy: The early development of television viewing", *Developmental Review,* vol. 30, pp. 239-255. DOI:10.1016/j.dr.2010.03.004

ANDERSON, J. R. (1987): "Skill Acquisition: Compilation of Weak-Method Problem Solutions", *Psychological Review*, vol. 94, n° 2, pp. 192-210.

ARBLASTER, A. (1982): "The importance of human factors in the design and use of computer languages", *International Journal of Man-Machine Studies,* vol. 17, pp. 211-224.

ARMSTRONG, A. y CASEMENT, C. (2000): *The Child and the Machine: How Computers Put our Children's Education at Risk,* Beltsville, Robins Lane Press.

ASPRAY, W. (1990): "Epilog", en W. Aspray (ed.): *Computing Before Computers,* Iowa State University Press, pp. 251-256.

AUGÉ, M. (2007): "Sobremodernidad. Del mundo de hoy al mundo de mañana", Contrastes, vol. 47). Disponible en: https://dialnet.unirioja.es/servlet/articulo?codigo=2244321.

AURELIANO, V., TEDESCO, P. y CASPERSEN, M. (2016): "Learning programming through stepwise self-explanations" *Information Systems and Technologies (CISTI)*, 2016 11th Iberian Conference.

AVRAM, A. (2014): "IDC Study: How Many Software Developers Are Out There?", QCon. Software Development Conference, 31 de enero. Disponible en: https://www.infoq.com/news/2014/01/IDC-software-developers.

BACHELARD, G. (1938): "La noción de obstáculo epistemológico", *La formación del espíritu científico,* Siglo XXI.

BALTIMORE, D. (2002): How biology became an information science, Nueva York, McGraw-Hill.

BATESON, G. (1972): "La epistemología de la cibernética", *Pasos hacia una ecología de la mente,* Buenos Aires, Lohlé-Lumen, pp. 345-350.

BAUMAN, Z. (2000): *Modernidad líquida,* Buenos Aires, Fondo de Cultura Económica.

BELL, T., ROSAMOND, F. y CASEY, N. (2012): "Computer Science Unplugged", en H. L. Bodlaender, R. Downey, F. V. Fomin y D. Marx (eds.): *The Multivariate Algorithmic Revolution and Beyond. Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday,* Berlín, Springer, pp. 398-456. DOI: 10.1007/978-3-642-30891-8.

BENASAYAG, M. y SMITH, G. (2010): "La crisis dentro de la crisis", *Las pasiones tristes. Sufrimiento psíquico y crisis social,* Buenos Aires, Siglo XXI.

BOWERS, C. A. (2000): *Let Them Eat Data: How Computers Affect Education, Cultural Diversity, and the Prospects of Ecological Sustainability,* Georgia, University of Georgia Athens.

BUCKINGHAM, D. (2008): *Más allá de la tecnología. Aprendizaje infantil en la era de la cultura digital,* Buenos Aires, Manantial.

CARRETERO, M. (2008): "El desarrollo del razonamiento y el pensamiento formal", en M. Carretero y M. Asensio (eds.): *Psicología del pensamiento,* Madrid, Alianza, pp. 36-58.

CARRETERO, M. y ASENSIO, M. (2008): *Psicología del Pensamiento,* Madrid, Alianza.

CARRETERO, M. y RODRIGUEZ MONEO, M. (2008): "Ideas previas, cambio conceptual y razonamiento", en M. Carretero y M. Asensio (eds.): *Psicología del pensamiento,* Madrid, Alianza.

CAVAROZZI, M. (2002): *Autoritarismo y democracia,* Buenos Aires, Eudeba.

CHENG, P. W. y HOLYOAK, K. J. (1989): "On the natural selection of reasoning theories", *Cognition,* vol. 33, pp. 285-313.

CHENG, P. y HOLYOAK, K. (1985): "Pragmatic Reasoning Schemas", *Cognitive Psychology,* vol. 17, pp. 391-416.

CHENG, P., HOLYOAK, K., NISBETT, R. y OLIVER, L. (1986): "Pragmatic versus Syntactic Approaches to Training Deductive Reasoning", *Cognitive Psychology,* vol. 18, pp. 293-328.

CHI, M., BASSOK, M., LEWIS, M., REIMANN, P. y GLASER, R. (1989). "Self Explanations: How Students Study and Use Examples in Learning to Solve Problems", *Cognitive Science,* vol. 13, pp. 145-182.

CHRISTAKIS, D. y ZIMMERMAN, F. (2006): "Media as a Public Health Issue", *Archives of Pediatrics & Adolescent Medicine,* vol. 160, n° 4, pp. 445-446. DOI: 10.1001/archpedi.160.4.445

CHRISTAKIS, D., EBEL, B. E., RIVARA, F. P. y ZIMMERMAN, F. (2004): "Television, Video and Computer Game Usage in Children Under 11 Years of Age", *Journal of Pediatrics,* vol. 145, n° 5, pp. 652-656.

COSMIDES, L. (1989): "The logic of social exchange: Has natural selection shaped how humans reason? Studies with the Wason selection task", *Cognition,* vol. 31, pp. 187-276.

COSTA, J. M. y MIRANDA, G. L. (2016): "Relation between Alice *software* and programming learning: a systematic review of the literature and meta-analysis", *British Journal of Educational Technology,* vol. 0, n° 0. DOI: 10.1111/bjet.12496.

DAHL, O. J. y DIJKSTRA, E. W. (1972): *Structured Programming.* Londres, Academic Press Ltd.

DENNING, P. J. (1984): "The Science of Computing: What Is Computer Science?", *American Scientist,* vol. 73, n° 1, pp. 16-19.

DENNING, P. J. (2010): *The Great Principles of Computing. American Scientist,* vol. 98, pp. 369-372.

DENNING, P. J. (2017): "Computational Thinking in Science", *American Scientist,* vol. 105, n° 1, p. 13.

FELLOWS, M. (1991): "Computer science in the elementary schools", en N. Fisher, H. Keynes y P. Wagreich (eds.): *Proceedings of the Mathematicians and Education Reform Workshop,* pp. 143 -163.

FELLOWS, M. (1993a): "Computer SCIENCE and Mathematics in the Elementary Schools", en N. D. Fisher, H. B. Keynes y P. D. Wagreich (eds.): *Mathematicians and Education Reform 1990-1991. Conference Board of the Mathematical Sciences, Issues in Mathematics Education,* 3, pp. 143-163.

FELLOWS, M. (1993b): "SIGACT trying to get children excited about CS", *Computing Research News,* vol. 5, n° 1, p. 7.

FELLOWS, M. y PARBERRY, I. (1993): "SIGACT trying to get children excited about CS", *Computing Research News,* vol. 5, n° 1, p. 7.

FUNDACIÓN SADOSKY (2013): CC 2016. Disponible en: http://www.fundacionsadosky.org.ar/wp-content/uploads/2014/06/cc-2016.pdf.

GUZDIAL, M. (2004): "Programming Environments for Novices", en S. Fincher, y M. Petre (eds.): *Computer science education research,* Lisse, Taylor & Francis, pp. 127-154.

HAASE, S., CHEN, H. L., SHEPPARD, S., KOLMOS, A. y MEJLGAARD, N. (2013): "What Does It Take to Become a Good Engineer? Identifying Cross-National

Engineering Student Profiles According to Perceived Importance of Skills", *International Journal of Engineering Education,* vol. 29, n° 3, pp. 698–713.

HOC, J. M., GREEN, T. R., SAMURÇAY, R. y GILMORE, D. J. (1990): "Part 1: Theoretical and Methodological Issues", *Psychology of Programming,* Londres, Academic Press.

HOLYOAK, K. y MORRISON, R. (2005): *The Cambridge Handbook of Thinking and Reasoning,* Nueva York, Cambridge University Press.

HOPPER, M. (1990): "Rattling SABRE—New Ways to Compete on Information", *Harvard Business Review.* Disponible en: https://hbr.org/1990/05/rattling-sabre-new-ways-to-compete-on-information.

INHELDER, B. y PIAGET, J. (1955): *De la lógica del niño a la lógica del adolescente. Ensayo sobre la construcción de las estructuras operatorias formales,* Barcelona, Paidós.

JENKINS, T. (2001): *The Motivation of Students of Programming,* tesis, University of Canterbury.

JENKINS, T. (2002): "On The Difficulty of Learning to Program", *Proceedings of the 3rd AnnualConference of the LTSN Centre for Information and Computer Sciences,* 4, pp. 53-58.

JENKINS, T. y DAVY, J. (2002): "Diversity and Motivation in Introductory Programming", *Innovation in Teaching and Learning in Information and Computer Sciences,* vol. 1, n° 1, pp. 1-9.

JOHNSON-LAIRD, P. (1975): *Mental Models. Towards a Cognitive Science on Language,* Inference and Consciousness, Cambridge University Press.

KASTENS, K. y LIBEN, L. (2007): "Eliciting Self-Explanations Improves Children's Performance on a Field-Based Map Skills Task", *Cognition and Instruction,* vol. 25, n° 1, pp. 45–74.

KNUTH, D. (1974): "Structured Programming with go to Statements", *Computing Surveys,* vol. 6, n° 4.

KNUTH, D. (1975): "Computer Science and its relation to Mathematics", *The American Mathematical Monthly,* vol. 81, pp. 323-343.

KNUTH, D. (1993): *C. L. Interview,* 7 de diciembre. Disponible en: http://tex.loria.fr/historique/interviews/knuth-clb1993.html.

KOLMOS, A. (2006): "Future Engineering Skills, Knowledge, and Identity", *Engineering Science, Skills, and Bildung,* pp. 165-185.

KOLMOS, A., DAHMS, M. y DU, X. (2010): "Transformation in Engineering Education", en UNESCO: *Engineering: Issues, Challenges and Opportunities for Development,* París, UNESCO Publishing, pp. 337-349.

KOSTYRKA-ALLCHORNE, K., COOPER, N. R. y SIMPSON, A. (2017): "The relationship between television exposure and children's cognition and behaviour: A systematic review", *Developmental Review,* vol. 44, pp. 19-58.

KRAMER, D. (1983): "Post-Formal Operations? A Need for Further Conceptualization", *Human Development,* vol. 26, pp. 91-105.

LAUFER, R. (1998): "¿El fin del trabajo? Desocupación, 'revolución tecnológica' y vigencia de las ideologías", *La marea. Revista de cultura, arte e ideas.* Disponible en: http://filo.uba.ar/contenidos/carreras/letras/catedras/historiasocialgeneral_b/sitio/sitio/fichafintrabajo.pdf

MANKINS, M. (2004): "Stop wasting valuable time", *Harvard Business Review,* vol. 82, n° 9, pp. 60-65.

MANKINS, M. C. y STEELE, R. (2006): "Stop making plans; start making decisions", *Harvard Business Review,* enero, pp. 76-84.

MANKINS, M. y GARTON, E. (2017): *Time, Talent, Energy. An Organization's Productive Power and How to Unleash It,* Harvard Business Review Press.

MARTIN, B. (2014): *My Lawn.* Disponible en: http://blog.cleancoder.com/uncle-bob/2014/06/20/MyLawn.html.

MARTIN, B. (2014): *Solid Principles of Object Oriented & Agile Design, Yale School of Management.* Disponible en: https://www.youtube.com/watch?v=TMuno5RZNeE.

MASTROGIUSEPPE, S. y IWANOW, M. (1992): *Programación I,* Rosario, UNR.

MITCHELL, N., MERTZ, K. y RYANT, R. (1994): "Learning Through Self-Explanation of Mathematics Examples: Effects of Cognitive Load", *Annual Meeting of the American Educational Research Association.*

NEWELL, A. y SIMON, H. (1970): "Human Problem Solving", *American Psychologist,* vol. 26, n° 2, pp. 145-159.

OECD. (2015): *Students, Computers and Learning: Making the Connection,* París, OECD Publishing.

PAPERT, S. (1980): *Mindstorms,* Nueva York, Basic Books.

PAPERT, S. (1992): *The Children's Machine: Rethinking School in the Age ot the Computer,* Nueva York, Basic Books.

PAPERT, S. (1993): *Mindstorms: Children, Computers, and Powerful Ideas, The Perseus Books Group.*

PENNINGTON, N. y GRABOWSKI, B. (1990): "The Tasks of Programming", *Psychology of Programming, Academic Press,* pp. 45-62.

PÉREZ ECHEVERRÍA, M. P. (2008): "Solución de Problemas", en M. Carretero y M. Asensio (eds.): *Psicología del Pensamiento,* Madrid, Alianza, pp. 199-218.

POLYA, G. (2014): *How to solve it. A New Aspect of Mathematical Method,* Princeton University Press.

POWERS, S. (2013): "Media and Technology in the Lives of Infants and Toddlers", *Journal of Zero to three: National Center for Infants, toddlers, and Families,* vol. 33, n° 4.

PRENSKY, M. (2001): "Digital Natives, Digital Immigrants", *On the Horizon,* vol. 9, n° 5.

PRESSMAN, R. (2010): *Ingeniería del Software,* México DF, McGraw Hill.

RAPAPORT, W. J. (2015): *Philosophy of Computer Science,* The State University of New York.

RENKL, A. (1997): "Learning from Worked-Out Examples: A Study on Individual Differences", *Cognitive SCience,* vol. 21, n° 1, pp. 1-29.

RIDEOUT, V. (2013): *Zero to Eight. Children's Media Use in America 2013. A Common Sense Media Research Study, Common Sense Media.* Disponible en: https://www.commonsensemedia.org/research/zero-to-eight-childrens-media-use-in-america-2013#.

RIDEOUT, V. J., FOEHR, U. G. y ROBERTS, D. F. (2010): *Generation M [superscript 2]: Media in the Lives of 8- to 18-Year-Olds, California: Henry J. Kaiser Family Foundation.* Disponible en: http://files.eric.ed.gov/fulltext/ED527859.pdf.

RIVIÈRE, Á. (1991): *Objetos con mente,* Madrid, Alianza.

RIVIÈRE, Á. (1999): "Desarrollo y educación: El papel de la educación en el "diseño" del desarrollo humano", en M. Belinchón, A. Rosa, M. Sotillo, y I. Marichalar: *Ángel Rivière. Obras Escogidas,* vol. III, Buenos Aires, pp. 203-242.

SAUSSURE, F. (1976): "Capítulo III: Objeto de la Lingüística", *Curso de lingüística general,* Buenos Aires, Centro Editor de América Latina.

SCHNOTZ, W. y BAADTE, C. (2014): *Surface and deep structures in graphics comprehension,* Alemania, Psychonomic Society. DOI: 10.3758/s13421-014-0490-2.

SCHNOTZ, W., & KULHAVY, R. (1994): *Comprehension of graphics,* Elsevier Science.

SIBILIA, P. (2005): *El hombre postorgánico. Cuerpo, subjetividad y tecnologías digitales,* Buenos Aires, Fondo de Cultura Económica.

TORR, J. D. (2003): "Computer-Assisted Education May Not Enhance Learning", Computers and Education. Disponible en: http://ic.galegroup.com.ezproxy.auckland.ac.nz/ic/ovic/ViewpointsDetailsPage/Vie.

TURKLE, S. y PAPERT, S. (1990): "Epistemological Pluralism: Styles and Voices within the Computer Culture", *Signs,* vol. 16, n° 1, pp. 128-157.

UNESCO (2010): *Engineering: Issues, Challenges and Opportunities for Development,* UNESCO Publishing.

UNESCO (2011): UNESCO Engineering Initiative. Disponible en: http://www.unesco.org/new/en/natural-sciences/science-technology/engineering/unesco-engineering-initiative/.

VYGOTSKI, L. S. (1978): *El desarrollo de los procesos psicológicos superiores,* Barcelona, Crítica.

WASSINK, J., SPIEGEL, K., y WASSINK, J. (2000): *Denken Als Discipline, Nederland.* Disponible en: https://www.cs.utexas.edu/users/EWD/video-audio/NoorderlichtVideo.html.

WERTSCH, J. (1988): *Vygotsky y la formación social de la mente,* Barcelona, Paidós.

WING, J. (2006): "Computational Thinking" communications of the ACM, vol. 49, n° 3, pp. 33-35.

WING, J. (2009): "Computational thinking and thinking", *Philosophical Transactions of the Royal Society,* vol. 366, pp. 3717–3725.

WIRTH, N. (1976): *Algorithms and Data Structures,* Canada, Pearson Education.

Cómo citar este artículo

D'ANGELO, V. S. (2018): "La programación de ordenadores. Reflexiones sobre la necesidad de un abordaje interdisciplinar", *Revista Iberoamericana de Ciencia, Tecnología y Sociedad -CTS,* vol. 13, n° 39, pp. 111-141.