EDSGER WYBE DIJKSTRA

Edgar Serna Montoya
Grupo de investigación SISCO. Funlam, Colombia
edgar.sernamo@amigo.edu.co

(Artículo de TRADUCCIÓN) (Recibido el 18 de febrero de 2009. Aceptado el 14 de mayo de 2009)

Rotterdam, Mayo 11 de 1930 – Nuenen, Agosto 6 de 2002

Fue uno de los más influyentes miembros de la generación fundadora de las ciencias computacionales. Son notables son aportes científicos en las áreas de:

- Diseño de Algoritmos
- Lenguajes de programación
- Diseño de programas
- Sistemas operativos
- Procesamiento distribuido
- Especificación y verificación formal
- Diseño de argumentos matemáticos

Además, fue admirable su incesante deseo por enseñar y por lograr la integración entre la ciencia computacional académica y la industria del software. En sus casi cuarenta años como científico computacional influenció a ambos sectores, y sus contribuciones le merecieron innumerables premios y reconocimientos, incluyendo el más alto honor de las ciencias computacionales, el ACM Turing Award en 1972.

Estudió física teórica en la Universidad de Leiden. Trabajó como investigador para Burroughs Corporation a principios de los años 1970. En la Universidad de Texas en Austin, Estados Unidos, ocupó el Schlumberger Centennial Chair in Computer Sciences. Se retiró en 2000.

Entre sus contribuciones a la informática están "el algoritmo de caminos mínimos", también conocido como "Algoritmo de Dijkstra", y haber sido el impulsor de un nuevo paradigma para la época, "la programación estructurada". Era conocido por su baja opinión de la sentencia GOTO en programación, que culminó en 1968 con su artículo "*Go To Statement Considered Harmful*", visto como un paso importante hacia el rechazo de esta expresión y de su eficaz reemplazo por estructuras de control como el bucle *while*. Era un aficionado bien conocido de Algol60 y trabajó en el equipo que desarrolló el primer compilador para este lenguaje; creó el primer sistema operativo con estructura jerárquica de niveles o capas, denominado "THE" - Technische Hogeschool, Eindhoven- que se utilizó con fines didácticos.

Desde los años 70, su principal interés fue la verificación formal. La opinión que prevalecía entonces era que primero se debe escribir un programa y seguidamente proporcionar una prueba matemática de su corrección. Dijkstra objetó que las pruebas que resultan son largas e incómodas, y que la prueba no da ninguna comprensión de cómo se desarrolló el programa. Diseñó un método alternativo al que llamó "la derivación de programas", según el cual se deben

"desarrollar prueba y programa conjuntamente": se comienza con una especificación matemática de lo que se supone que el programa va a hacer, luego se aplican transformaciones matemáticas a la especificación hasta que se transforma en un programa que se pueda ejecutar. El programa resultante se determina correcto por construcción. Muchos de sus últimos trabajos tratan sobre las maneras de hacer fluida la argumentación matemática. Dijkstra murió el 6 de agosto de 2002 después de una larga lucha contra el cáncer.

La siguiente es la traducción al español de su artículo: "*On the cruelty of really teaching computing science*" - EWD1036, de 1988.

ACERCA DE LA CRUELDAD DE REALMENTE ENSEÑAR CIENCIAS DE LA COMPUTACIÓN

Edsger Wybe Dijkstra

Esta charla trata de algunas de las consecuencias científicas y educacionales provenientes de la idea de que los computadores representan una novedad radical. Para darle contenidos claros a tal presunción, tenemos que ser mucho más precisos acerca de lo que se quiere decir en este contexto con el uso del adjetivo "radical". Lo haremos en la primera parte de esta charla, en la cual vamos a proveer evidencia que respalde esta suposición.

La manera usual con la cual se planifica hoy para el mañana es en el vocabulario de ayer. Lo hacemos porque tratamos de avanzar con los conceptos que nos son familiares, los cuales adquieren un significado en nuestra experiencia pasada. Por supuesto, las palabras y los conceptos no encajan precisamente porque nuestro futuro difiere de nuestro pasado, pero las estiramos un poco. Los lingüistas están bastante familiarizados con el fenómeno en el cual los significados de las palabras evolucionan en el tiempo, pero también saben que éste es un proceso lento y progresivo.

Es el método más común cuando se trata de lidiar con la novedad: utilizar metáforas y analogías para tratar de vincular lo nuevo con lo viejo, lo novedoso con lo familiar. En un proceso suficientemente lento y gradual, esto funciona razonablemente bien; en el caso de una discontinuidad aguda, sin embargo, el método colapsa: aunque podemos glorificarlo con el nombre "sentido común", nuestra experiencia pasada ya no es más relevante, las analogías se tornan muy superficiales, y las metáforas se hacen engañosas en vez de reveladoras. Esta es la situación que caracteriza a la novedad "radical".

Lidiar con una novedad radical requiere un método ortogonal. Debemos considerar su propio pasado, las experiencias recogidas y los hábitos formados en él como un desafortunado accidente de la historia, y debemos acercarnos a la novedad radical con la mente en blanco, rechazando conscientemente el intento de vincularla con lo que ya nos es familiar, debido a que lo familiar es desesperanzadamente inadecuado. Debemos, con una especie de personalidad dividida, tomar una novedad radical como algo desasociado por propio derecho. Comprender una novedad radical implica crear y aprender un lenguaje extraño, el cual no puede ser traducido a nuestra lengua materna -Cualquiera que haya aprendido mecánica cuántica sabe a lo que me refiero. No hace falta decirlo, ajustarse a las novedades radicales no es una actividad muy popular, ya que requiere mucho trabajo. Por la misma razón, las novedades radicales no son por sí mismas bienvenidas.

A esta altura, bien se pueden preguntar por qué he prestado tanta atención y gastado tanta elocuencia en una noción tan simple y obvia como una novedad radical. Mi razón es muy simple: las novedades radicales son tan perturbadoras que tienden a ser suprimidas o ignoradas, al punto que la mera posibilidad de su existencia generalmente se niega antes que admitirla.

Voy a ser breve en cuanto a la evidencia histórica. Carl Friedrich Gauss, el Príncipe de los Matemáticos -pero algo cobarde-, sin duda estaba al tanto del destino de Galileo- y, probablemente, podría haber predicho las

acusaciones a Einstein– cuando decidió suprimir su descubrimiento de la geometría no Euclidiana, dejando que Bolyai y Lobatchewsky recibieran las críticas. Es probablemente más revelador ir un poco más atrás, a la Edad Media, en la que era característico pensar que "razonar mediante analogías" era descontrolado; otra característica era el total estancamiento intelectual, y ahora vemos porque ambas características van juntas.

Una razón para mencionar esto es resaltar que, si se desarrolla un oído entrenado para las analogías no garantizadas, uno puede detectar gran cantidad de pensamiento medieval hoy. La otra cosa que no puedo resaltar lo suficiente es que la fracción de la población, para la cual el cambio gradual parece ser cualquier cosa menos el único paradigma de la historia, es muy grande, probablemente mucho más grande de lo que se esperaría. Ciertamente cuando comencé a observarlo, su número resultó ser mucho mayor de lo que esperaba. Por ejemplo, la gran mayoría de la comunidad matemática nunca confrontó la suposición tácita de que hacer matemáticas continúa básicamente como el mismo tipo de actividad mental que siempre fue: los nuevos temas vendrán, florecerán y se irán como lo hicieron en el pasado; pero siendo lo que es el cerebro humano y nuestras formas de enseñar y aprender, el entendimiento, las matemáticas, la resolución de problemas y el descubrimiento matemático continuarán siendo básicamente lo mismo.

Herbert Robbins expone claramente por qué descarta un salto cuántico en la habilidad matemática:

> *Nadie va a correr 100 metros en cinco segundos, sin importar cuánto se invierta en entrenamiento y máquinas. Lo mismo puede decirse acerca del uso del cerebro. La mente humana no es diferente ahora de lo que era hace cinco mil años. Y cuando se trata de matemáticas, hay que darse cuenta que se trata de utilizar la mente humana al límite de su capacidad.*

Mi comentario en el margen fue "*¡entonces reduzca el uso del cerebro y calcule!*". Usando la propia analogía de Robbins, se puede resaltar que, para ir rápido desde A hasta B, pueden existir alternativas a la de correr, que son órdenes más efectivas. Robbins rechaza de plano analizar cualquier alternativa al valioso uso del cerebro, llamado "*hacer matemáticas*", alejando así el peligro de la novedad radical mediante el simple método de ajustar sus definiciones a sus necesidades: simplemente por definición, las matemáticas continuarán siendo lo que solían ser. Demasiado para los matemáticos.

Déjenme darles un ejemplo más de la desconfianza generalizada acerca de la existencia de las novedades radicales y, por consiguiente, de la necesidad de aprender cómo tratar con ellas. Es el accionar educacional que prevalece, para el cual el cambio, casi imperceptible, parece ser el paradigma exclusivo. ¡Cuántos textos educacionales no son recomendados porque apelan a la intuición del estudiante! Constantemente tratan de presentar todo aquello que podría ser una emocionante novedad, como les sea posible, como algo muy familiar. Conscientemente tratan de vincular el material nuevo con lo que se supone es el mundo familiar del estudiante. Por ejemplo al enseñar aritmética: en lugar de enseñar 2 + 3 = 5, el horrendo operador aritmético "más" es cuidadosamente disfrazado llamándolo "y", y a los pequeños niños se les presentan primero varios ejemplos familiares, con objetos claramente visibles como manzanas y peras, en contraste al uso de objetos numerables como porcentajes y electrones, que no lo son. La misma tonta tradición se refleja en la universidad en diferentes cursos introductorios de cálculo para los futuros físicos, arquitectos o economistas, en los que cada uno los adorna con ejemplos de sus respectivos campos.

El dogma educacional parece ser que todo está bien siempre y cuando el estudiante no se dé cuenta que está aprendiendo algo verdaderamente nuevo; generalmente, el presentimiento del estudiante es de hecho correcto. Considero como un serio obstáculo la falencia de una práctica educativa en preparar a la próxima generación para el fenómeno de novedades radicales. Cuando el Rey Fernando visitó la conservadora universidad de Cervera, el Rector orgullosamente le dijo al monarca: "*Lejos esté de nosotros, Señor, la peligrosa novedad de pensar*". Los problemas de España en el siglo que siguió justifican mi caracterización del problema como "serio". Demasiado para

adoptarse en la educación el paradigma de cambio gradual.

El concepto de novedades radicales es de importancia contemporánea ya que, mientras estamos mal preparados para tratar con ellas, la ciencia y la tecnología no se muestran expertas en influirlas sobre nosotros. Ejemplos científicos antiguos son la teoría de la relatividad y la mecánica cuántica; ejemplos tecnológicos modernos son la bomba atómica y la píldora. Durante décadas, los dieron lugar a un torrente de corrientes religiosas, científicas o cuasi-científicas. Día a día podemos observar el profundo error de enfoque con el cual los últimos se abordan, ya sea por nuestros políticos y líderes religiosos o por el público en general. Demasiado para el daño hecho a nuestra paz mental por las novedades radicales.

Traje esto a colación debido a mi convencimiento de que las computadoras automáticas representan una novedad radical y de que sólo identificándolas como tal podemos identificar todo lo irrelevante, los conceptos errados y la mitología que las rodea. Una inspección más a fondo revelará que esto es todavía peor, ya que que las computadoras automáticas engloban no sólo una novedad radical sino dos de ellas. La primera es una consecuencia directa del poder bruto de las computadoras actuales. Todos sabemos cómo tratar con algo tan grande y complejo: divide y vencerás; por ejemplo, vemos el todo como una composición de partes y tratamos con las partes por separado, y si una parte es muy grande, repetimos el procedimiento: la ciudad se estructura en barrios, que a su vez se estructuran en calles, que contienen edificios, que están hechos de paredes y pisos, que están construidas de ladrillos..., eventualmente se llega a las partículas elementales. Y tenemos a todos nuestros especialistas sobre el tema, desde el ingeniero civil, el arquitecto y el físico de estado sólido y demás. Ya que, en cierto sentido, el todo es más grande que sus partes, la profundidad de una descomposición jerárquica es algún tipo de logaritmo del cociente entre los tamaños del todo y las partes más pequeñas.

Desde un bit a cien mega bytes, desde un microsegundo a media hora de cómputos, confrontamos un cociente completamente abrumador de 10^9. El programador está en la desigual posición en la que la suya es la única disciplina y profesión donde un cociente tan gigante, lo cual completamente sobrepasa nuestra imaginación, debe consolidarse por una sola tecnología. Debe poder pensar en términos de jerarquías conceptuales que son mucho más profundas que todas aquellas que debió enfrentar una sola mente alguna vez. Comparado con ese número de niveles semánticos, la teoría matemática promedio es casi plana.

Evocando la necesidad de profundas jerarquías conceptuales, la computadora automática nos confronta con un radical desafío intelectual que no tiene precedente histórico. Nuevamente, debo enfatizar esta novedad radical ya que el verdadero creyente en el cambio gradual y las mejoras incrementales no puede verla. Para él, una computadora automática es algo como una familiar caja registradora, sólo que algo más grande, rápida y más flexible. Pero la analogía es ridículamente superficial: en órdenes de magnitud es como comparar un medio de transporte como el avión supersónico, con un bebé que gatea, ya que el cociente de velocidad es sólo de mil.

La segunda novedad radical es que la computadora automática es nuestro primer dispositivo digital de gran escala. Tuvimos un par de notables componentes discretos: acabo de mencionar la caja registradora y podemos agregar la máquina de escribir con sus teclas individuales, con un sólo golpe podemos escribir una Q o una W pero, aunque las teclas están una al lado de la otra, nunca se mezclan las dos. Pero tales mecanismos son la excepción, y la amplia mayoría de nuestros mecanismos son vistos como dispositivos analógicos cuyo comportamiento sobre un amplio rango es una función continua de todos los parámetros involucrados: si presionamos la punta del lápiz un poco más fuerte, obtenemos una línea levemente más gruesa; si el violinista ubica su dedo levemente fuera de su posición correcta, reproduce una nota levemente desafinada. A esto debería agregar que, al punto que nos vemos como mecanismos, nos vemos primordialmente como dispositivos analógicos: si nos esforzamos un poco más esperamos rendir un poco más. A menudo el comportamiento no es solamente una función

continua sino también monótona, ya que para ver si un martillo es adecuado sobre un cierto rango de clavos, lo probamos con el más pequeño y el más grande de los clavos del rango, y si el resultado de ambos experimentos es positivo, estamos perfectamente predispuestos a creer que el martillo será apropiado para todos los clavos intermedios.

Es posible, inclusive tentador, ver un programa como un mecanismo abstracto, como alguna clase de dispositivo. Pero hacerlo, sin embargo, es altamente peligroso, ya que la analogía es muy superficial debido a que un programa es, como mecanismo, totalmente diferente de todos los familiares dispositivos analógicos con los cuáles crecimos. Como toda la información digitalizada, tiene la inevitable e incómoda propiedad de que la menor de las posibles perturbaciones –cambios a un sólo bit– puede tener las más drásticas consecuencias. Para complementar agrego que la situación no cambia en su esencia por la introducción de la redundancia o la corrección de errores. En el mundo discreto de la computación, no hay métrica significativa en la cual pequeños cambios y pequeños efectos vayan de la mano, y nunca los habrá. Esta segunda novedad radical comparte el destino usual a todas las novedades radicales: es negada, porque su verdad sería demasiado incómoda. No tengo idea lo que esta negación y descreencia específica le cuesta a los Estados Unidos, pero un millón de dólares al día parece una modesta estimación.

Luego de describir, en los términos más amplios posibles, lo admito, la naturaleza de las novedades computacionales, debo ahora proveer la evidencia de que tales novedades son, de hecho, radicales. Lo haré explicando una serie de fenómenos que de otra manera serían extraños por la frustrante ocultación o negación de su aterradora extrañeza. Cierta cantidad de estos fenómenos se agrupan bajo el nombre de "*Ingeniería de Software*". Así como la economía es conocida como "*La ciencia miserable*", la ingeniería de software debería ser conocida como "*La disciplina condenada*"; condenada porque ni siquiera puede acercarse a su meta, dado que la misma es en sí misma contradictoria. La ingeniería de software, por supuesto, se presenta a sí misma como otra causa valiosa, pero no es así, ya que si lee cuidadosamente su literatura y analiza lo que realmente hacen quienes se avocan a ella, se descubrirá que la ingeniería de software adopta como estatuto el "*Cómo programar si usted no puede*".

La popularidad de su nombre es suficiente para hacerla sospechosa. En lo que denominamos sociedades primitivas, la superstición de que conocer el verdadero nombre de alguien otorgaba un poder mágico sobre él, no es inusual. Difícilmente somos menos primitivos: ¿por qué persistimos en contestar el teléfono con el poco útil "aló" en vez de nuestro nombre? Tampoco estamos por encima de la primitiva superstición de que podemos tener cierto control sobre algún demonio malicioso desconocido llamándolo por un nombre seguro, familiar e inocente, tal como ingeniería. Pero esto es totalmente simbólico, así como demostró uno de los fabricantes de computadoras de los EE.UU. hace unos años, cuando contrató a cientos de nuevos "*ingenieros de software*" mediante el simple mecanismo de elevar a todos sus programadores a ese exaltante rango. Demasiado para ese término.

La práctica está impregnada de la confortable ilusión de que los programas son simplemente dispositivos como cualquier otro, la única diferencia que se admite es que su fabricación pueden requerir un nuevo tipo de especialistas: los programadores. Desde allí hay sólo un pequeño paso hasta medir la “productividad del programador” en términos de la “cantidad de líneas producidas por mes”. Esta es una unidad de medida muy costosa, porque anima a escribir código insípido, y si la idea es contar líneas de código, no deben verse como “líneas producidas”, sino como “líneas gastadas”. El sentido común actual es tan tonto como el hecho de contabilizar una cuenta del lado erróneo del balance.

Además de la noción de productividad, el control de calidad también se distorsiona por la confortable ilusión de que los programas funcionan como lo hacen otros aparatos. Han pasado ya dos décadas desde que se señaló que el *testing* de programas puede convincentemente demostrar la presencia de errores, pero nunca puede demostrar su ausencia. Después de citar devotamente este comentario bien publicitado, el ingeniero de

software vuelve al orden del día y continúa refinando sus estrategias de *testing*, tal como el alquimista de antaño, quien refinaba sus purificaciones crisocósmicas.

Un profundo malentendido revela el término "mantenimiento de software" como resultado del cual muchas personas creen que los programas –inclusive los mismísimos lenguajes de programación– están sujetos a desgaste y ruptura. Es famosa la historia de la empresa petrolera que creía que sus programas en PASCAL no durarían tanto como sus programas en FORTRAN "porque PASCAL no tenía soporte". En el mismo sentido también llama la atención la sorprendente facilidad con que se acepta la sugerencia de que los males de la producción de software de deben, en gran medida, a la falta de "herramientas de programación" apropiadas –no falta sino que aparezca la frase "banco de trabajo del programador". Nuevamente, lo miope de la analogía subyacente se debe a la Edad Media: las confrontaciones con las insípidas "herramientas" del tipo de "animación de algoritmos" con el que actualmente se quiere comparar al desarrollo de software, recuerda el proceso del negocio de aceite de serpientes de aquella época.

Finalmente, para corregir la posible impresión de que la inhabilidad de enfrentar la novedad radical está confinada al mundo industrial, analicemos la actual popularidad de la Inteligencia Artificial. Se esperaría que la gente se sintiera aterrorizada por los "cerebros gigantes de máquinas que piensan"; de hecho, el atemorizante computador se vuelve menos atemorizante si se utiliza solamente para simular un no-computador más familiar. Con toda seguridad que esta explicación será controvertida por bastante tiempo dado que la Inteligencia Artificial, como imitadora de la mente humana, prefiere verse a sí misma como a la vanguardia, mientras que la anterior explicación la relega a la retaguardia. "El esfuerzo de utilizar máquinas para imitar la mente humana siempre me ha parecido bastante tonto: las usaría para imitar algo mejor".

Hasta aquí la evidencia de que las novedades computacionales son, de hecho, radicales. Ahora viene la segunda –y más difícil– parte de esta charla: las consecuencias educativas y científicas de lo ya expuesta. Las consecuencias educativas son, por supuesto, las más engorrosas, por lo tanto se discutirán luego; primero tratemos con las ciencias de la computación en sí mismas. ¿Qué es la computación? ¿De qué trata la ciencia de la computación? Una vez que todo está dicho y hecho, la única cosa que los computadores pueden hacer por nosotros es manipular símbolos y producir resultados de tales manipulaciones. De las observaciones previas, es necesario recordar que este es un mundo discreto y, más aún, tanto los números como los símbolos involucrados y la cantidad de manipulaciones realizadas, son órdenes de magnitudes mayores a los que el hombre puede concebir: desconciertan totalmente cualquier imaginación y por lo tanto no es conveniente siquiera tratar de imaginarlas.

Pero antes que un computador esté listo para realizar alguna clase de manipulación con sentido –o cálculo, si se prefiere– es necesario escribir un programa. ¿Qué es un programa? Varias respuestas son posibles: se puede ver como lo que transforma un computador de propósito general en un manipulador de símbolos de propósito específico, y lo hace sin necesidad de cambiar un solo cable; lo que fue una enorme mejora respecto de las máquinas con paneles de cables dependientes del problema. Es mejor describirlo de la otra manera: un programa es un manipulador de símbolos abstracto, que puede convertirse en uno concreto suministrándole un computador. Después de todo, el propósito de los programas ya no es más instruir a las máquinas, en estos días, el propósito de las máquinas es ejecutar los programas.

Por lo tanto, es necesario diseñar manipuladores de símbolos abstractos. Que se vean como programas o, para usar una terminología más general, usualmente fórmulas de algún sistema formal un tanto elaboradas. Realmente ayuda ver a un programa como una fórmula: primero, pone la tarea del programador en la perspectiva correcta, tiene que derivar esa fórmula; segundo, explica por qué el mundo de las matemáticas ignora el desafío de la programación, los programas eran fórmulas mucho más largas que las usuales, al punto que ni siquiera las reconocieron como tales. Ahora, de vuelta al trabajo del programador: tiene que derivar esa fórmula, tiene que

derivar ese programa, y se conoce una única forma confiable de hacerlo, mediante la manipulación de símbolos.

Ahora el círculo está cerrado: se construyen manipuladores de símbolos mecánicos mediante la manipulación de símbolos humanos. Por lo tanto, la ciencia de la computación está –y siempre estará– relacionada con la interacción entre la manipulación de símbolos mecanizada y humana, usualmente llamadas "computación" y "programación", respectivamente. Un beneficio inmediato de esta visión es que revela a la "programación automática" como una contradicción de términos; y un beneficio posterior es que indica dónde ubicar la ciencia de la computación en el mapa de las disciplinas intelectuales: en la dirección de la matemática formal y la lógica aplicada, pero mucho más allá de donde se encuentra actualmente, dado que la ciencia de la computación se interesa en el uso efectivo de los métodos formales en una escala mucho, mucho mayor de la que el ser humano ha sido testigos hasta ahora.

Dado que ningún emprendimiento es respetable por estos días sin una Sigla de Tres Letras STL, propongamos que se adopte, para la ciencia de la computación, la sigla IMF -Iniciativa de los Métodos Formales-, y, para estar del lado seguro, es mejor seguir los brillantes ejemplos de nuestros líderes y hacer de ella una Marca Registrada. En el largo plazo se espera que la ciencia de la computación trascienda a sus disciplinas padres, matemática y lógica, de forma efectiva y realizando una parte significativa del sueño de Leibniz: proveer un cálculo simbólico como una alternativa al razonamiento humano. Nótese la diferencia entre "imitar" y "proveer" una alternativa.

De más está decir que esta visión acerca de lo qué trata la ciencia de la computación no es universalmente aplaudida. Por el contrario, encuentra oposición –a veces hasta violenta– desde todo tipo de direcciones. Mencionemos como ejemplos:

1. La comunidad matemática, que quisiera continuar creyendo que el Sueño de Leibniz es una ilusión irreal
2. La comunidad empresarial, quienes, habiéndoseles vendido la idea de que los computadores harían la vida más simple, no están mentalmente preparados para aceptar que sólo resolvieron los problemas más simples a precio de crear uno mucho más difícil
3. La subcultura del programador compulsivo, cuya ética prescribe que una idea tonta y un mes de codificación frenética, deberían bastar para hacerlo millonario de por vida
4. Los ingenieros en computación, quienes quisieran continuar actuando como si fuera solamente cuestión de mayor flujo de bits o más *flops* por segundo
5. Los militares, quienes están hoy totalmente absorbidos por el negocio de usar computadores para transformar partidas de miles de millones de dólares en la ilusión de seguridad automática
6. Todo tipo de ciencias para las cuales la computación ahora actúa de alguna especie de refugio interdisciplinario
7. El negocio educativo que siente que, si tiene que enseñar matemática formal a los estudiantes de Ciencias de la Computación, también deberían cerrar sus facultades.

Con este último ejemplo se alcanza, imperceptiblemente pero también inevitablemente, la parte más engorrosa de esta charla: consecuencias educativas. El problema con la política educativa es que es difícilmente influenciada por consideraciones científicas derivadas de los tópicos dictados, y casi completamente determinada por circunstancias ajenas a la ciencia, como las expectativas conjugadas de los estudiantes, sus padres y sus futuros empleadores, y el enfoque prevaleciente del rol de la universidad. ¿El acento está en formar sus graduados para los trabajos de nivel inicial de hoy o en proveer a sus alumnos con el bagaje intelectual y las actitudes que perduraran por otros 50 años? ¿Se le da rencorosamente a las ciencias abstractas solo un rincón lejano en el campus, o se reconocen como el motor indispensable de la industria de alta tecnología? Aún si se hace lo último, ¿se reconoce una industria de alta tecnología como tal si su tecnología pertenece principalmente a las matemáticas formales? ¿Proveen las universidades a la sociedad el liderazgo intelectual que necesita o sólo el entrenamiento que demanda?

La retórica académica tradicional está perfectamente dispuesta a dar a estas cuestiones las respuestas tranquilizadoras, pero es difícil creerlas. A modo de ilustración, en un artículo de David Flaherty (Flaherty, 1988), groseramente se establece que "*Además, la élite de los negocios descarta a los académicos e intelectuales como ampliamente irrelevantes e impotentes*". Así, si se mira en una borrosa bola de cristal hacia el futuro de la educación en ciencias de la computación, sobrecogedoramente se ve la deprimente imagen del "negocio acostumbrado". A las universidades les seguirá faltando el coraje de enseñar ciencia dura, continuarán orientando mal a los estudiantes, y cada nuevo escalón de infantilización del currículum será exaltado como progreso educativo.

Este ejercicio de observar en la borrosa bola de cristal, ofrece predicciones invariablemente melancólicas y usualmente correctas, a las que es necesario acostumbrarse, y que no sean impedimentos para hacer sugerencias, aunque sea meramente un ejercicio vano cuyo único efecto es hacerlos sentir culpables. Por ejemplo, se puede comenzar por limpiar el lenguaje con que se expresan los términos computacionales: denominar a un *bug* como error y no como *bug -bicho, fastidio, sabandija* (Nota del traductor). Es mucho más honesto porque pone manifiestamente la culpa donde corresponde, es decir, en el programador que cometió el error. La metáfora animada del *bug* que se introdujo maliciosamente mientras el programador no estaba mirando es intelectualmente deshonesta, ya que disfraza el hecho de que el error es propia creación del programador. Lo agradable de este simple cambio de vocabulario es que tiene un profundo efecto: antes, un programa con sólo un error solía ser "casi correcto", después de ello un programa con un error es simplemente "erróneo" -porque tiene un error.
Otra sugerencia lingüística, un poco más rigurosa, es confrontar el síndrome de "si-este-tipo-quiere-hablarle-a-ese-tipo": nunca se refieran a partes de programas o piezas de equipo en una terminología antropomórfica, ni permitan hacerlo a sus estudiantes. Esta mejora lingüística es mucho más difícil de implementar de lo que podrían pensar: si se considerara la introducción de multas para las violaciones, digamos veinticinco centavos para estudiantes de grado, cincuenta centavos para estudiantes de postgrado y cinco dólares para miembros de la facultad, para final del primer semestre del nuevo régimen, se habrá recolectado suficiente dinero para dos becas.

La razón para esta última sugerencia es que la metáfora antropomórfica –por cuya introducción se puede culpar al mismo John von Neumann– es una enorme desventaja para cada comunidad informática que la adoptó. Se encuentran programas que quieren cosas, saben cosas, esperan cosas, creen cosas, etc., y cada vez eso genera confusiones evitables. La analogía que subyace a esta personificación es tan superficial que no es solamente engañosa sino también paralizante. Es engañosa en el sentido que sugiere que se puede lidiar con el desconocido discreto en términos del familiar continuo, es decir, nosotros mismos, *quod non*. Es paralizante en el sentido que, debido a que las personas existen y actúan en el tiempo, su adopción efectivamente impide un despegue de la semántica operacional y lleva a las personas a pensar sobre los programas en términos de comportamientos computacionales, basados en un modelo computacional subyacente. Esto es malo, porque el razonamiento operacional es un tremendo desperdicio de esfuerzo mental.

Vamos a explicar la naturaleza de ese tremendo desperdicio y que la frase "tremendo desperdicio de esfuerzo mental" no es una exageración. Es necesario asumir un lenguaje altamente técnico: es el tipo de matemáticas que se puede hacer con las manos en los bolsillos. El punto a comunicar es que si se tiene que demostrar algo respecto de todos los elementos de un conjunto grande, es desesperanzadamente ineficiente tratar con todos los elementos del conjunto individualmente: el argumento eficiente no se refiere a elementos individuales en lo absoluto y se lleva a cabo en términos de la definición del conjunto. Consideremos la figura plana Q, definida como el cuadrado de 8 por 8 del cual, en dos esquinas opuestas, se quitan dos cuadrados de 1 por 1. El área de Q es 62, que equivale al área combinada de 31 cuadros de 1 por 2. El teorema es que la figura Q no puede ser cubierta por 31 de estos cuadros. Otra manera de exponer el teorema es que si se

utiliza papel cuadriculado y se cubre éste ubicando cada siguiente cuadro en dos nuevos recuadros adyacentes, ninguna distribución de 31 cuadros dará como resultado la figura Q. Así, una posible manera de probar el teorema es generar todas las posibles distribuciones de cuadros y verificar para cada distribución que no da como resultado la figura Q.

El argumento simple, sin embargo, es como sigue: pintar los recuadros del papel cuadriculado como un tablero de ajedrez. Cada cuadro, cubriendo dos recuadros adyacentes, cubre 1 recuadro blanco y 1 negro y, por consiguiente, cada distribución cubre tantos recuadros blancos como recuadros negros. En la figura Q, sin embargo, el número de recuadros blancos y el número de recuadros negros difiere en 2 –esquinas opuestas sobre la misma diagonal– y por consiguiente ninguna disposición de cuadros da como resultado la figura Q.

No sólo es el simple argumento anterior muchas órdenes de magnitud más corto que la investigación exhaustiva de las posibles distribuciones de 31 cuadros, es también esencialmente más poderoso, dado que cubre la generalización de Q reemplazando el cuadrado original de 8 por 8 por cualquier rectángulo con lados de longitud par. Siendo infinito el número de tales rectángulos, el método previo de exploración exhaustiva es esencialmente inadecuado para probar nuestro teorema generalizado. Y esto concluye el ejemplo. Se presentó porque ilustra fácilmente el poder de las matemáticas terrenales; no hace falta decirlo, la negación de explotar este poder de las matemáticas terrenales escala al suicidio intelectual y tecnológico. La moraleja de la historia es: tratar con todos los elementos de un conjunto ignorándolos y trabajando con la definición del conjunto.

Volvamos a la programación. La aseveración de que un programa dado cumple una cierta especificación, escala a una aseveración sobre todos los cálculos computacionales que podrían ocurrir bajo el control de ese programa dado. Y dado que este conjunto de cálculos está definido por el programa dado, la anterior moraleja dice: trate con todas los cálculos posibles bajo control de un programa dado ignorándolas y trabajando con el programa. Debemos aprender a trabajar con el texto de los programas mientras –temporalmente– se ignora que admiten la interpretación de código ejecutable.

Otra manera de decir la misma cosa: un lenguaje de programación, con su sintaxis formal y las reglas de demostración que definen su semántica, es un sistema formal para el cual la ejecución del programa provee solamente un modelo. Es bien conocido que los sistemas formales deberían ser tratados por derecho propio, y no en términos de un modelo específico. Nuevamente, el corolario es que se debe razonar sobre los programas sin siquiera mencionar su posible "comportamiento". Hasta aquí esta excursión técnica en el motivo por el cual el razonamiento operacional sobre la programación es "un tremendo desperdicio de esfuerzo mental" y por qué, en consecuencia, en la ciencia de la computación debería prohibirse la metáfora antropomórfica.

Un programa no es más que la mitad de una conjetura, la otra mitad es la especificación funcional que se supone que satisface el programa. La tarea del programador es presentar las conjeturas completas como teoremas demostrados.

Consideremos la siguiente forma de hacer justicia a las novedades radicales de la computación en un curso introductorio a la programación: por un lado, se enseña algo que se parece al cálculo de predicados, pero se hace de manera muy distinta a los filósofos. A fin de entrenar al programador novato en la manipulación de fórmulas sin interpretar, se enseña más como álgebra booleana, familiarizando al estudiante con todas las propiedades algebraicas de los conectivos lógicos. Para romper aún más los vínculos con la intuición, ser renombran los valores {verdadero, falso} del dominio booleano como {negro, blanco}. Por otro lado, se enseña un lenguaje de programación imperativo simple y claro, con un *skip* y una asignación múltiple como sentencias básicas, con una estructura de bloque para variables locales, el punto y coma como operador para composición de sentencias, una bonita construcción alternativa, una bonita repetición y, si se quiere, una llamada a procedimiento. A esto se agrega un mínimo de tipos de datos, digamos booleanos, enteros, caracteres y cadenas. Lo esencial es

que, para lo que sea que se introduzca, la semántica correspondiente está definida por las reglas de demostración que la acompañan. Desde el comienzo, y a través de todo el curso, se enfatiza en que la tarea del programador no es sólo escribir un programa, sino que su tarea principal es dar una prueba formal de que el programa que propone cumple la especificación funcional - igualmente formal. Mientras se diseñan demostraciones y programas conjuntamente, el estudiante adquiere una amplia oportunidad de perfeccionar su destreza manipulativa con el cálculo de predicados. Finalmente, para hacer llegar el mensaje de que este curso introductorio a la programación es principalmente un curso en matemáticas formales, nos encargamos de que el lenguaje de programación en cuestión no haya sido implementado en el campus de manera que los estudiantes estén protegidos de la tentación de probar sus programas.

Esta es una propuesta seria, y sumamente sensible. Su única desventaja es que es demasiado radical para muchos, quienes, siendo incapaces de aceptarla, se ven forzados a inventar alguna justificación rápida para desestimarla, no importa lo inválida sea. Veamos unas pocas de esas justificaciones.

No necesitan tomar esta propuesta seriamente porque es tan ridícula que está obviamente desconectada del mundo real. Pero esa cometa no va a volar, ya que el mundo real es otra cosa: los problemas del mundo real son principalmente aquellos con los que ustedes se quedan después de negarse a aplicar sus efectivas soluciones. Así que, probemos de nuevo. No necesitan tomar seriamente esta propuesta porque es sumamente surrealista intentar enseñar tal material a alumnos de primer año. ¿No sería esa una salida fácil? Acaban de postular que esto sería por lejos muy difícil. Pero esa cometa tampoco va a volar, dado que el postulado se demostró falso hace tiempo: desde principios de los 80, se dio este curso introductorio a la programación a cientos de estudiantes de primer curso de grado cada año -ya que decir esto no es suficiente, la sentencia previa debería repetirse por lo menos otras dos veces. Así que, intentemos de nuevo. Admitiendo renuentemente que podría quizás enseñarse a estudiantes suficientemente dóciles, todavía rechazan esta propuesta porque tal curso se desviaría tanto de lo que los estudiantes de 18 años están habituados y esperan, que enseñarles esto es un acto de irresponsabilidad educativa: sólo frustraría a los estudiantes. No hace falta decirlo, esa cometa tampoco va a volar. Es cierto que el estudiante que nunca ha manipulado fórmulas sin interpretar se da rápidamente cuenta que se confronta con algo totalmente distinto a cualquier cosa que haya visto antes. Pero afortunadamente, las reglas de manipulación son en este caso tan pocas y simples que muy poco después hace el excitante descubrimiento de que comienza a dominar el uso de una herramienta que, en toda su simplicidad, le da un poder que sobrepasa sus sueños más audaces.

Enseñar a jóvenes desprevenidos el uso efectivo de los métodos formales es uno de los placeres de la vida, porque es extremadamente gratificante. En pocos meses, encuentran su camino en un mundo nuevo con justificado grado de confianza, que es radicalmente novedoso para ellos; en pocos meses, su concepto de cultura intelectual adquiere una dimensión radicalmente novedosa, ¿no es esto de lo de lo que se trata la educación. Las universidades no deberían temerle a enseñar novedades radicales; por el contrario, es su llamado dar la bienvenida a la oportunidad de hacerlo. Su disposición a hacerlo es nuestra principal salvaguarda contra las dictaduras, sean del proletariado, del *establishment* académico, o de la élite corporativa.

FUENTES

1. E. W. Dijkstra. (1989). On the cruelty of really teaching computing science. Communications of the ACM, Vol. 32, No. 12, pp. 1398-1404.
2. Flaherty, D. H. (1988). Who Rules Canada? In Stephen Richards Graubard's "In search of Canada", pp. 99-128.
3. Wikipedia.org

Ω