Formando programadores para el futuro[*]

Maximiliano Contieri[**]

Resumen

El mercado actual busca programadores para apilar ladrillos utilizando sus lenguajes de moda como señala Raevskiy (2020). Dichos profesionales son formados en herramientas específicas que tienen ciclos de vida muy cortos y luego se vuelven obsoletos. Actualmente existen alternativas para realizar desarrollos informáticos utilizando gente inteligente, capaz de tomar sus propias decisiones y participar en el proceso creativo. Los programadores del futuro deberán ser excelentes y declarativos como afirma van Lamsweerde (2001). Profesionales que conozcan de algoritmos y complejidad y que sepan diseñar sistemas y encajar funcionalidades nuevas en soluciones preexistentes.

Palabras claves: Software. Desarrollo de software. Programación. Ciencia de Datos. Ingeniería de software. Educación.

Abstract

Today's market is looking for programmers to stack bricks using their fad languages. These professionals are trained in specific tools that have very short life cycles and then rapidly become obsolete. Currently, there are alternatives to carry out developments using intelligent people, capable of making their own decisions and taking part in the creative process. The programmers of the future must be excellent and declarative. Professionals who know not only algorithms and software complexity, but also how to design systems and fit new functionality into existing solutions.

Keywords: Software. Software Development. Programming. Data Science. Software Engineering. Education.

[*] Enviado: 02-02-2021. Aceptado: 08-04-2021.

[**] Licenciado en Ciencias de la Computación por la Universidad de Buenos Aires. Docente en la cátedra de Ingeniería de Software. Desarrollador de Software con 25 años de trayectoria profesional. Correo electrónico: mcontieri@dc.uba.ar

Resumo

O mercado de hoje está procurando programadores para empilhar tijolos usando suas linguagens da moda, como Raevskiy (2020) aponta. Esses profissionais são treinados em ferramentas específicas que têm ciclos de vida muito curtos e, depois, se tornam obsoletas. Atualmente existem alternativas para realizar o desenvolvimento de computadores utilizando pessoas inteligentes, capazes de tomar suas próprias decisões e participar do processo criativo. Os programadores do futuro devem ser excelentes e declarativos, como afirma van Lamsweerde (2001). Profissionais que conhecem algoritmos e complexidade e sabem projetar sistemas e encaixar novas funcionalidades nas soluções existentes.

Palavras-chave: Software. Desenvolvimento de software. Programação. Ciência de dados. Engenharia de software. Educação.

Introducción

La mayoría de los desarrolladores del mundo están realizando cambios en sistemas antiguos o deben utilizar complejas librerías preexistentes o *frameworks* sobre los que tienen poco control y escasas posibilidades de modificación tal como señalan Putryc y Kark (2008). En la industria del software de hoy en día es muy improbable encontrarse con la necesidad de hacer un desarrollo desde cero, con una página completamente en blanco. La vida útil de un programador específico envejece junto con el lenguaje de moda. Este período de tiempo suele ser menor a una década, por lo que la industria descarta por obsoletos a los profesionales antes de diez años de haberse formado como afirma Glass (2000).

En las últimas décadas estuvieron de moda como supuestas balas de plata según la definición de Brooks (1987), lenguajes como Visual Basic, C++, Java, Php, Perl, Javascript, Ruby, Python y GoLang.

Algunos de ellos ya están dejando de utilizarse dejando su lugar a nuevas modas como releva Combs (2019). Los conceptos generales (que Frederick Brooks denomina esenciales como detalla Contieri (2020) se absorben mejor en nuestra primera etapa académica. Debemos focalizar en formar estos conceptos de modo que los profesionales puedan migrar fácilmente de una tecnología (que Brooks denomina accidental) hacia otra tecnología (también accidental y, por lo tanto, rápidamente obsoleta).

La amenaza

Para 2021 ya existen numerosas alternativas de inteligencia artificial y *machine learning*, capaces de realizar tareas de programación de bajo nivel y algorítmicas como señalan Arteaga (2017) y Gutsztat (2020).

Las máquinas virtuales modernas (aquellas que interpretan el lenguaje de alto nivel en el que programamos y que existen, por ejemplo, en los navegadores web) optimizan el código por nosotros como detallan Yunfa, Wanqing y Congfeng (2010). A medida que la tecnología avance, ya no se requerirán programadores de bajo nivel del mismo modo que nadie busca bibliotecarios hoy en día como afirma Aprendizbiblios (2019). Similarmente a lo que sucede con otras profesiones automatizables y obsoletas según concluyen Molina, Benítez y Ernst (2018).

Nuestra profesión se sigue basando en enseñar programación imperativa de bajo nivel como si estuviéramos ante los problemas de los años 1960 o 1970 donde el hardware especializado eran las tarjetas perforadas y las máquinas de cinta abierta.

Educamos a nuestros estudiantes para que realicen optimizaciones algorítmicas absurdas y obsoletas, muy cerca del lenguaje de las máquinas y muy lejos del modelado de entidades del problema que estamos representando en nuestra solución informática. Formamos profesionales para entender cómo funciona una computadora y rebajarse a hablar en su idioma, formando modelos mentales de bajo nivel y tratando de "razonar" del modo algorítmico en que procesa una *máquina de Turing* (1948). Sin embargo, hoy en día, podemos elaborar modelos semánticos varias capas de abstracción más arriba a medida que nos acercamos a los modelos mentales declarativos nos alejamos al mismo tiempo de las abstracciones computables de las máquinas. Los lenguajes de programación más declarativos nos permiten razonar e inferir reglas sobre nuestros modelos. A contramano de dicha tendencia, nuestros alumnos escriben sus instrucciones basándose en unos y ceros.

La ley de Moore (1965) nos habla de la velocidad del cambio del hardware y de su crecimiento exponencial en capacidad de procesamiento. Con respecto al software, crecemos linealmente limitados por realidades de hace muchas décadas y la inercia de los formadores. En analogía con la teoría económica de Malthus (1798) necesitamos alcanzar a la curva del hardware para poder liberar todo nuestro potencial antes que las máquinas lo hagan por nosotros.

La falta de abstracción y entendimiento de la realidad es un síntoma de una mentalidad inercial, basada en las restricciones de tiempo de procesador y almacenamiento de los años 60

y 70. Dichas restricciones quedan limitadas en la actualidad a dominios muy específicos y no son representativas de los problemas que tenemos que resolver hoy en día como ingenieros de software en la mayoría de nuestros trabajos como afirma Otero (2012).

La alternativa

Necesitamos formar generalistas, pero no teóricos. Gente que entienda las bases y fundamentos de la ingeniería de software y se pueda adaptar a implementaciones accidentales basadas en las herramientas actuales, pero también a las del futuro que aun hoy no conocemos. Conceptos como el diseño de software, modelos de ciclo de vida, trabajo en equipo y construcción de una teoría compartida (17), integración y despliegue continuos o arquitectura, son mucho más importantes que aprender a minimizar el uso de CPU, a usar *React*, el framework *Vue* o la librería *Keta*.

"Es muy difícil realizar predicciones, especialmente sobre el futuro". Esta frase, atribuida al genial Niels Bohr, nos indica que deberíamos tener cuidado al intentar anticiparnos al futuro. El Turing Award 2004 Alan Kay (2003) dijo: "La mejor forma de predecir el futuro es inventarlo". No sabemos a ciencia cierta cuál será el camino de automatización de las máquinas, pero podemos predecir que la parte más creativa e ingenieril de nuestra profesión debería ser uno de los últimos bastiones en caer ante la automatización. Las próximas generaciones de desarrolladores de software deberán focalizar sus habilidades en el diseño, el modelado del mundo real y la creación de abstracciones que evolucionen junto con el dominio del problema para evitar construir software obsoleto y heredado como afirma Feathers (2002).

Este enfoque estratégico no se limita únicamente las carreras de desarrollo. En ciencia de datos existen problemas similares: científicos de datos focalizados en optimizar algoritmos existentes en vez de formar profesionales generalistas, capaces de entender los problemas académicos y comerciales de nuestro país con una batería de soluciones y buen criterio para determinar cuál elegir ante cada situación. Deberían entrenarse para modelar el problema real y evaluar distintas soluciones posibles.

Los profesionales de ciencias de datos cuentan con una increíble variedad de herramientas para ajustar sus modelos. Según Fjelland (2020) todavía no estamos cerca de encontrar el segundo “súper algoritmo” capaz de resolver problemas específicos con una solución genérica. Nuestros futuros profesionales cuentan con la única computadora de propósito general que puede resolver de manera decente diversos problemas específicos: sus mentes brillantes.

Nuestra responsabilidad es alimentar y estimular esos cerebros con problemas reales y no con soluciones de juguete que las computadoras resuelven (ya en 2020) de manera mucho más eficiente. Paradojamente, malgastar nuestros cerebros pidiéndoles que realicen optimizaciones de bajo nivel para las máquinas es la peor forma de optimizar nuestro recurso más costoso.

Históricamente, hemos privilegiado el desarrollo de herramientas teóricas y muy específicas. Esto es muy común y deseable en la ciencia porque los avances teóricos suelen preceder en décadas a las implementaciones y usos concretos. Sin embargo, en el desarrollo de software, los descubrimientos y hallazgos se encuentran mayormente en la industria privada antes que en la academia. Nuestros profesionales deben estar formados en metodologías y conceptos por encima de tecnologías y lenguajes de moda accidentales. Esto genera una tensión entre el mercado que desea "implementadores expertos en una herramienta" para descartarlos cuando ésta cumpla su ciclo de moda de 5 o 10 años. Nuestros profesionales no deberían ser desechables ni reciclables. Debemos formarlos en técnicas y ellos deben mantenerse actualizados constantemente, como ocurre en otras profesiones como la medicina, la física o la bio-tecnología.

¿Qué necesitamos enseñar?

Además de habilidades "blandas" de construcción y trabajo en equipo (ya que el software surge de una actividad colectiva como afirma Naur (1985), debemos enseñar técnicas de diseño y prototipación para validar nuestras soluciones de alto nivel. En cuanto al software, es imperioso enseñar diseño de soluciones, focalizando en el comportamiento de nuestros modelos y, parafraseando a Donald Knuth (2014), el autor histórico de la mayoría de los algoritmos que utilizamos hoy en día, evitando optimizaciones prematuras por querer jugar un juego que las máquinas dominan mucho mejor que nosotros.

La oportunidad

Formar talento es una opción accesible para cualquier país con buen nivel académico, como Argentina. Invertir en formar excelentes ingenieros de software es una decisión estratégica y una oportunidad de despegue que ya han aprovechado muchos otros países como Estonia, Irlanda, Israel o la India. En esta línea se encuentra trabajando actualmente la fundación Sadosky (2021).

En Argentina tenemos excelentes docentes, buen nivel de inglés, inmejorable huso horario para dialogar con los Estados Unidos y Europa, y una cultura compatible con los países más desarrollados. Necesitamos priorizar las tecnologías de la información y, dentro de ellas, formar ingenieros inteligentes y declarativos por sobre programadores mecanizados y optimizadores de bajo nivel.

¿Qué les debemos enseñar a nuestros ingenieros?

Nuestros profesionales deberán tener conocimientos básicos de programación, algoritmos, complejidad y bases de datos. Por sobre todo, deben aprender a realizar diseños basados en modelos de integración continua y despliegue continuo, con pruebas automatizadas, utilizando técnicas ágiles como Desarrollo Guiado por Pruebas creado por Beck (2012).

El software producido debe ser declarativo y basado en el comportamiento deseado (y especificado en las pruebas funcionales automatizadas); debemos dejar de pensar en el paradigma reinante en los 60 y 70, basado en tipos de datos y manipulaciones de archivos y cadenas de texto, para enfocarnos en modelos de alto nivel que acompañen la simulación de cualquier aspecto del mundo real que deseemos representar para resolver un problema determinado como nos enseña West (2004).

Las técnicas de diseño basado en comportamiento son agnósticas con respecto a la tecnología accidental de moda y esto permite que un ingeniero formado con estos conceptos hace 30 años pueda realizar desarrollos concretos aún hoy. Lamentablemente, dicha situación no se ve replicada por programadores que dominaron algún lenguaje de moda, que ya prácticamente no tiene utilización y eso hace que no encuentren buenas opciones en el mercado laboral. Se da la paradoja de que un oficio con pleno empleo descarte estos profesionales por no haberse podido adaptar tal como muestra Dudkin (2020).

El cambio, actualmente, es aún más vertiginoso. Las tecnologías duran mucho menos y la obsolescencia nos pisa los talones, a menos que seamos inteligentes y amplios y tengamos la formación apropiada.

Trabajo Futuro

Este es un artículo de opinión. Como trabajo futuro para apoyar la presente tesis deberíamos realizar un trabajo cuantitativo incluyendo cifras de rotación de empleados tomando la información de Openqube (2021), tiempo promedio en cada trabajo según la edad y los

estudios realizados, entre otras cosas. Para ello debemos utilizar técnicas relacionadas a las ciencias sociales bajo un enfoque multidisciplinario.

Conclusiones

El futuro ya llegó. No tenemos mucha idea de cómo será el trabajo de un programador en 5 o 10 años, pero tenemos indicios fuertes de que no estará relacionado con escribir algoritmos basados en estructuras de datos. Debemos formar profesionales que entiendan rápidamente un problema de la vida real y sepan construir simuladores con saltos conceptuales muy pequeños, para que puedan evolucionar acompañando los problemas que estamos resolviendo hoy.

Bibliografía

Aprendizbiblios (2019). "La revolución de las máquinas. Adiós al bibliotecario". Disponible en: https://aprendizbibliotecologa.wordpress.com/2019/02/20/la-revolucion-de-las-maquinas-adios-al-bibliotecario/ [Fecha de consulta: 28/05/2021].

Arteaga, S. (2017). "DeepCoder, la inteligencia artificial de Microsoft que crea programas". Disponible en: https://computerhoy.com/noticias/software/deepcoder-inteligencia-artificial-microsoft-que-crea-programas-58774 [Fecha de consulta: 28/05/2021].

Beck, K. (2012). "Why does Kent Beck refer to the 'rediscovery' of test-driven development?" Disponible en: https://www.quora.com/Why-does-Kent-Beck-refer-to-the-rediscovery-of-test-driven-development-Whats-the-history-of-test-driven-development-before-Kent-Becks-rediscovery [Fecha de consulta: 28/05/2021].

Brooks, F. (1987). "No Silver Bullet Essence and Accidents of Software Engineering". *Computer*, 20 (4): 10-19.

Combs, V. (2019). "Java and JavaScript dominated software development in the 2010s". Disponible en: https://www.techrepublic.com/article/java-and-javascript-dominated-software-development-in-the-2010s/ [Fecha de consulta: 28/05/2021].

Contieri, M. (2020). "No hay balas de plata". Disponible en: https://medium.com/dise%C3%B1o-de-software/no-hay-balas-de-plata-b16449eb79c5 [Fecha de consulta: 28/05/2021].

Dudkin, I. (2020). "Programming languages to avoid". Disponible en: https://www.itproportal.com/features/programming-languages-to-avoid/ [Fecha de consulta: 28/05/2021].

Feathers, M. (2002). *Working Effectively With Legacy Code*. Object Mentor, Inc.

Fjelland, R. (2020). "Why general artificial intelligence will not be realized". *Humanities and Social Sciences Communications*, 7 (1): 1-9.

Glass, R. L. (2000). "Practical programmer: On personal technical obsolescence". *Communications of the ACM*, 43 (7): 15-17.

Gutsztat, L. (2020). "Haldo Sponton: 'Con Augmented Coding vamos a patear el tablero de la industria del software'". Disponible en: https://stayrelevant.globant.com/es/haldo-sponton-con-augmented-coding-vamos-a-patear-el-tablero-de-la-industria-del-software/ [Fecha de consulta: 28/05/2021].

Kay, A. (2003). "Turing Award". Disponible en: https://amturing.acm.org/award_winners/kay_3972189.cfm [Fecha de consulta: 28/05/2021].
Knuth, D. E. (2014). *Art of computer programming. Seminumerical algorithms*. Addison-Wesley Professional.
Malthus, T. (1798). *An Essay on the Principle of Population As It Affects the Future Improvement of Society, with Remarks on the Speculations of Mr. Goodwin, M. Condorcet and Other Writers. London: J. Johnson in St Paul's Church-yard.*
Molina, M., Benítez, N. y Ernst, C. (2018). "Cambios tecnológicos y laborales sus implicancias en el mercado de trabajo de Argentina". International Labour Organization.
Moore, G. *(1965 [2020]). "Cramming more components onto integrated circuits". Electronics Magazine*.
Naur, P. (1985). "Programming as theory building". *Microprocessing and Microprogramming*, 15 (5): 253-261.
Openqube (2021). "Resultados de la encuesta de sueldos 2021.01diciembre-enero". Disponible en: https://sueldos.openqube.io/encuesta-sueldos-2021.01/ [Fecha de consulta: 28/05/2021].
Otero, C. (2012). "Software Design Challenges". *IT Performance Improvement*. Disponible en: http://www.ittoday.info/ITPerformanceImprovement/Articles/2012-06Otero.html [Fecha de consulta: 28/05/2021].
Putrycz, E, Kark, A. (2008). Connecting Legacy Code, Business Rules and Documentation Recuperado de https://link.springer.com/chapter/10.1007/978-3-540-88808-6_5 (visitado el 28/05/2021).
Raevskiy, M. (2020). "Why Golang Is Bad for Smart Programmers". Disponible en: https://raevskymichail.medium.com/why-golang-bad-for-smart-programmers-4535fce4210c [Fecha de consulta: 28/05/2021].
Sadosky (2021). "Program.AR". Disponible en: http://www.fundacionsadosky.org.ar/programas/programar/ [Fecha de consulta: 28/05/2021].
Turing, A. (1948). "Intelligent Machinery (Report)". *National Physical Laboratory*: 3-4.
Van Lamsweerde, A. (2001). "Goal-Oriented Requirements Engineering: A Guided Tour". Disponible en: https://www.info.ucl.ac.be/~avl/files/RE01.pdf [Fecha de consulta: 28/05/2021].
West, D. (2004). *Object thinking. Pearson Education*. Microsoft.
Yunfa, L. y Wanqing, C. (2010). "A Survey of Virtual Machine System: Current Technology and Future Trends" (ponencia). *Electronic Commerce and Security, International Symposium*.