Aprender a programar

Por Mauro Gullino

Dos enfoques que permiten pensar la enseñanza de la programación. Cuál es mejor para cada perfil y su relación con los profesionales del diseño.

Cada vez más profesionales del diseño se ven involucrados en ese campo del conocimiento denominado (sintéticamente) «Programación». El ámbito web es quizá el más ilustrativo: en muy pocos casos podemos finalizar el trabajo sin recurrir al acto de «programar».[1] Esto se debe a que las interfaces se han vuelto muy complejas. Además (al menos en nuestra región) es casi imposible ofrecer únicamente servicios de diseño web: también debemos maquetar y eventualmente programar. Esta tendencia a la programación abarca muchas actividades, algo que se refleja en el florecimiento de sitios como Codecademy.com.[2]

Es importante recordar que el software (como producto de la actividad de programar) es un objeto de conocimiento en sí mismo. Por eso existen profesionales dedicados específicamente a este tema. Pero también se puede observar que no solo programan los especialistas, así como no solo diseñan los diseñadores profesionales.

Según nuestra experiencia en la enseñanza, los diseñadores aprenden a programar de forma distinta que los ingenieros o técnicos informáticos. Estas dos maneras de responder al conocimiento, resultan de características cognitivas que los estudiantes de estas carreras demuestran. El perfil general (real o deseado) es el siguiente:

- **Estudiantes de una carrera técnica:** mayor predisposición al planteo matemático y analítico. Los planes de estudio comienzan con materias básicas (álgebra, física, análisis matemático y lógica), para internalizarse de a poco en técnicas propias de cada profesión. La metodología suele incluir clases magistrales, cuadernos de ejercicios y exámenes escritos de acreditación.

- **Estudiantes de carreras de diseño:** mayor predisposición al abordaje proyectual, al «hacer» pragmático. Desde el comienzo se realizan trabajos similares a los reales, aumentando la complejidad y el grado de eficacia en la elaboración. La metodología es mayormente la de maestro-aprendiz y se consolida en la realización y entrega de trabajos prácticos.

Estos dos perfiles tienen su correlato en la enseñanza de la programación. Se denominan, respectivamente: *aproximación bottom-up* y *aproximación top-down*.

Bottom-up

Esta metodología enfatiza la progresión de conceptos. Supone que la definición de un concepto es imprescindible para el siguiente, por lo que el orden resulta clave. Por ejemplo, para comprender Bioquímica debemos entender Química Orgánica antes, y previo a ello Química Inorgánica. Los conceptos se apoyan unos sobre otros. Recién al final tendremos un panorama completo y la posibilidad de hacer algo «real».

Este enfoque es el más elegido en las carreras técnicas. Se comienza por conceptos como sintaxis y estructuras de control para demostrar cómo se escribe y piensa la programación. Se tratan estos temas en profundidad antes de pasar al siguiente. Los conceptos se afianzan con ejercicios, pero al contar con pocas herramientas, estos son (casi siempre) creados artificialmente con fines didácticos. Por esa razón, no hay demasiada relación entre lo ejercitado y la actividad profesional futura.

Con el paso del tiempo se agregan nuevas ideas, siempre haciendo foco en dominar cada etapa para así continuar con la siguiente. Esta metodología se basa en esa progresión, algo visible en los planes de estudio a través de la «correlatividad» de asignaturas y exámenes.

Esta visión tradicional es enciclopedista y conceptualiza al estudiante como un «libro en blanco», que se va completando con temas cada vez más difíciles. Un enfoque muy arraigado en la academia.

Top-down

Esta metodología es más pragmática: no trabaja tanto con la definición de conceptos y el orden entre ellos, sino poniendo el foco en el «hacer». Por medio de la demostración y el ejemplo el estudiante tendrá una visión de lo que significa programar. Primero se hace y luego se comprende; no se explica cada herramienta por separado para luego integrarlas en una solución, sino que se construye la solución y luego se la «□diseccciona». Dependiendo del tiempo con que se cuente (o el interés de los participantes), se puede profundizar en cada tema progresivamente. Podemos afirmar que se empieza por el final y luego se retrocede: a la inversa que en el *bottom-up*.

Este enfoque es el más elegido en carreras de diseño, en función de que el futuro profesional se nutra de vocabulario y conocimientos propios de la informática, sin que ello implique transformarse en una «enciclopedia viviente» sobre el tema. Además este enfoque se adapta mejor al estudiante de diseño, que verá en funcionamiento artefactos concretos como producto del proceso. Asimismo, el tiempo disponible es mucho menor.

Esta metodología da por sentado que, al principio, utilizaremos cosas sin saber cómo funcionan. Con el paso del tiempo comprenderemos más el por qué de cada comportamiento. El enfoque *bottom-up*, en cambio, omite los temas que no puede explicar y los reserva para examinarlos con detenimiento más adelante.

Un ejemplo conocido

Hace tiempo existe una campaña publicitaria (en tono cómico) de un sitio web para aprender inglés *online* (OpenEnglish.com). En ella, un estudiante «moderno» se burla de otro que representa al típico alumno de «instituto de inglés» en diversas situaciones. Estos dos abordajes ilustran el *top-down* y el *bottom-up*.

El estudiante tradicional se aproxima al inglés con una metodología *bottom-up*. Se lo muestra cargando libros, pronunciando mal y confundiendo significados. Paradójicamente, quizá su desempeño en un análisis sintáctico de oraciones sea muy bueno. En cambio, el estudiante del sistema *online*, pronuncia muy bien y comprende lo que se le dice. Él es un estudiante *top-down*. Quizá no pueda nombrar cuál es el tiempo verbal en uso, pero lo utiliza. Tal vez no sepa de memoria la lista de verbos irregulares en pasado, pero sabe utilizar los más habituales.

Debe notarse el obvio sesgo en favor de *top-down*, debido a los fines publicitarios. Sin embargo, la caracterización de las dos metodologías resulta bastante realista.

Qué metodología elegir

Por supuesto, ambas aproximaciones tienen pros y contras:

- **Ventajas de *bottom-up*:** existe amplio material con esta orientación y, aunque cuestionable, la historia avala su eficacia como formación a largo plazo. Es más fácil extrapolar conceptos a nuevos lenguajes y tecnologías.

- **Desventajas de *bottom-up*:** puede ser frustrante trabajar siempre con ejercicios *ad-hoc*. Riesgo de desmotivarse por no comprender «para qué me sirve» tal o cual cosa. El tiempo necesario para llegar a trabajar en casos reales es muy extenso.

- **Ventajas de *top-down*:** suele ser más estimulante porque se ven resultados rápidos. Es útil para mostrar panoramas generales y hacer divulgación.

- **Desventajas de *top-down*:** pueden formarse ideas equivocadas sobre el funcionamiento de algunos conceptos, por la falta de dedicación a cada uno de ellos. Se precisa una intervención más cercana al estudiante.

Para un ingeniero de software será preciso conocer en profundidad los temas fundamentales, por lo que el *bottom-up* puede ser más eficiente a largo plazo.[3] En cambio, para un diseñador que desea conocer cómo funciona la programación y modificar o escribir algunas porciones de código, lo más conveniente será el *top-down*.

Publicado el 05/05/2014

1. Tecnologías como LESS y SASS están incorporando ideas de la programación incluso a CSS.
2. Estas plataformas son claros ejemplos de lo que llamaremos metodología *top-down*, como se refleja en frases como *«Learn by Doing»* o *«Learn to make an iPhone game in an hour»*.
3. Se ha dicho que *bottom-up* focaliza en la construcción de las herramientas más que en lo que puede hacerse con las herramientas mismas. (Reek, Margaret. *«A top-down approach to teach programming»*. SIGCSE ☐95 3/95 Nashville, TN, USA)

FOROALFA
ISSN 1851-5606
https://foroalfa.org/articulos/aprender-a-programar