La prosa del código: programar para ser leído

Alejandro Umaña Miranda

Santiago Valverde Álvarez

Cuando se empieza en la carrera de ingeniería en computación, no suele decirse que aprender a programar se parece, en muchos sentidos, a aprender a escribir. Se enseña sintaxis, estructuras de datos, algoritmos, paradigmas, complejidad. Se evalúa si el programa compila, si resuelve el problema correcto, si pasa los casos de prueba, si usa una solución suficientemente eficiente. La pregunta por la escritura casi nunca aparece de forma explícita. Rara vez se pregunta si el código tiene ritmo, si está organizado de una manera que permita entenderlo con naturalidad, si comunica con claridad la intención de quien lo escribió. Y, sin embargo, conforme se avanza en la disciplina, se vuelve evidente que es importante. Programar no es solamente dar instrucciones a una máquina. Es también producir un texto técnico que otros seres humanos tendrán que leer, interpretar, cuestionar, corregir y continuar.

Esta relación entre programación y literatura no debe entenderse como una simple comparación estética, ni como una metáfora elaborada para embellecer el oficio del programador. Se trata de una semejanza concreta, visible en la práctica cotidiana del desarrollo de software. Basta con enfrentarse a un sistema heredado, a una base de código sin documentación o a un conjunto de funciones cuyos nombres no explican nada para notar que el problema no es solo técnico. Es también un problema de escritura. Hay código que se deja leer como un texto bien construido, donde cada parte anticipa la siguiente, donde la estructura orienta, donde los nombres acompañan la lógica. Y hay código que se resiste como un texto torpe, lleno de repeticiones, ambigüedades y saltos bruscos. En ambos casos, lo que cambia no es únicamente el funcionamiento, sino la experiencia de lectura.

Donald Knuth (1992), en su propuesta de *Literate Programming*, sostiene que quien practica la programación literaria puede verse como un ensayista cuya preocupación central es la exposición de ideas y la excelencia del estilo. Su planteamiento no consiste simplemente en comentar más el código o en adornarlo con explicaciones innecesarias. Lo que propone es reorganizar la forma en que se piensa

un programa, partiendo del hecho de que la comprensión humana no siempre sigue el mismo orden en que una máquina ejecuta instrucciones. Por eso, escribir bien un programa implica presentar sus conceptos en el orden que resulte más claro para el lector. Esta idea sigue siendo provocadora incluso hoy, porque obliga a desplazar el centro de la actividad: el programa no debe construirse primero para la máquina, sino para la persona que intentará entenderlo después.

Esa misma intuición aparece de manera aún más directa en Abelson y Sussman (1996), cuando afirman que un lenguaje de programación no es únicamente un medio para hacer que una computadora ejecute operaciones, sino también un medio formal para expresar ideas sobre metodología. En esa misma línea, señalan que los programas deben escribirse para que las personas los lean, y solo incidentalmente para que las máquinas los ejecuten. La frase se cita con frecuencia, quizá porque resume de manera precisa algo que muchos desarrolladores aprenden tarde: la legibilidad no es un detalle opcional ni una preocupación secundaria. Es parte de la calidad del software. Un programa puede producir la salida correcta y aun así estar mal escrito. Puede funcionar hoy y, al mismo tiempo, estar sembrando dificultades para mañana.

La semejanza entre programación y literatura puede observarse, en primer lugar, en la forma en que ambas construyen significado. No se trata simplemente de que las dos utilicen palabras o símbolos. La similitud es más profunda. Tanto en el texto literario como en el código, una unidad aislada rara vez significa mucho por sí sola. Su sentido depende del contexto en que aparece, de la red de relaciones que establece con otras unidades, del lugar que ocupa dentro de una estructura mayor. Un nombre de variable, una clase o una función adquieren verdadero significado por su relación con el resto del sistema. De manera similar, una palabra en un poema, en una novela o en un ensayo no puede separarse del tono, del ritmo y de las asociaciones que la rodean. En ambos casos, el contexto no es un accesorio. Es parte esencial del significado.

También existe una similitud en la noción de estilo. En literatura, el estilo no es solamente una cuestión ornamental. Es una forma de ordenar la experiencia, de decidir qué se muestra primero, qué se omite, qué se enfatiza, qué voz se adopta. En programación ocurre algo parecido. Dos personas pueden resolver correctamente el

mismo problema y producir programas profundamente distintos. Una puede escribir funciones cortas, nombres precisos y una secuencia lógica fácil de seguir. Otra puede concentrar demasiadas responsabilidades en un solo bloque, abusar de abreviaturas, repetir estructuras y obligar al lector a reconstruir mentalmente lo que el texto debió aclarar por sí mismo. En ambos casos hay estilo, pero no todo estilo comunica igual de bien. La cuestión no es eliminar la voz del programador, sino disciplinarla para que sea útil.

Esta observación es importante porque durante mucho tiempo la enseñanza de la programación ha privilegiado una visión instrumental del código. Se aprende a programar como si se tratara de ensamblar mecanismos: cada pieza debe cumplir una función, y el criterio final es que el sistema opere. Desde luego, esa dimensión es indispensable. Un programa que no funciona no puede defenderse por su belleza léxica. Pero reducir la programación a su ámbito funcional deja por fuera una parte decisiva del trabajo real. En la práctica profesional, programar no es solo producir soluciones desde cero. Es entrar en conversaciones técnicas ya empezadas. Es leer las decisiones de otros, descubrir supuestos que nadie documentó, interpretar convenciones, detectar contradicciones, corregir errores sin romper lo que ya existe. Es decir, gran parte del trabajo consiste en leer.

Ese hecho debería bastar para replantear la formación. Si el ingeniero pasa una porción significativa de su tiempo leyendo código ajeno, entonces la habilidad de escribir código legible no puede verse como un lujo estético. Es una condición de trabajo colaborativo. Un sistema bien escrito reduce el esfuerzo cognitivo de quienes participan en él. Disminuye el tiempo necesario para entender una parte del programa, facilita la detección de errores y hace menos costoso introducir cambios. Por el contrario, un código opaco incrementa la fricción en cada modificación. Obliga a interpretar demasiado, genera miedo a tocar ciertas partes y convierte tareas sencillas en procesos inciertos. Igual que ocurre con un texto mal redactado, la mala escritura en código no solo incomoda. Produce consecuencias concretas.

La conexión con la literatura también aparece en la idea de reescritura. Ningún escritor serio considera que la primera versión de un texto sea necesariamente la mejor. Escribir implica volver sobre lo escrito, cortar, reordenar, renombrar, simplificar,

precisar. El trabajo no termina cuando la frase “funciona”, sino cuando comunica lo que debe comunicar de la mejor manera posible. En programación ocurre algo muy semejante. Refactorizar no es un acto accesorio ni una manía de perfeccionismo. Es una forma de revisión textual. Consiste en volver al código para mejorar su estructura, sin alterar su comportamiento observable. Renombrar una función para que exprese mejor su intención, dividir un bloque demasiado largo, mover responsabilidades a un módulo más adecuado, eliminar duplicaciones. Todo eso se parece menos a reparar una máquina que a editar un texto.

Visto así, el programador no solo resuelve problemas. También toma decisiones narrativas. Decide qué mostrar primero y qué después. Decide cuándo encapsular un detalle y cuándo exponerlo. Decide cuánto contexto necesita el lector antes de enfrentar una abstracción. Decide si una función cuenta una sola idea o varias mezcladas. Decide si el recorrido por el sistema será claro o accidentado. En cierto sentido, cada programa propone una lectura. Hay programas que guían al lector con cuidado, como un ensayo bien construido. Otros lo arrojan a una secuencia de fragmentos inconexos, obligándolo a reconstruir una intención que debió estar más visible.

Incluso los comentarios de código pueden pensarse desde esta relación. Un mal comentario se parece a una mala nota al pie: repite lo obvio, interrumpe innecesariamente o explica mal aquello que el propio texto debería dejar claro. Un buen comentario, en cambio, no traduce línea por línea lo que ya se ve. Más bien aporta contexto, justifica una decisión, aclara una restricción de negocio, advierte sobre una consecuencia no evidente. En otras palabras, también aquí hay una dimensión de escritura responsable. No se trata de escribir más, sino de escribir mejor. Lo mismo vale para la documentación técnica: su propósito no es existir por obligación, sino reducir incertidumbre en quien llega después. No obstante, sí debería existir.

Escribir código ilegible no afecta solo a quien lo escribió. Afecta al equipo, al mantenimiento del sistema y, en última instancia, a las personas que dependen del software. Cuando una decisión mal explicada o una estructura innecesariamente compleja dificultan corregir un error, el costo puede trasladarse a usuarios, clientes, instituciones o servicios críticos. De ahí que la claridad no sea únicamente una virtud

estética. Es una forma de responsabilidad profesional. Un ingeniero no trabaja en aislamiento absoluto. Produce artefactos que circulan, que se comparten, que se heredan. Escribir con cuidado es reconocer esa realidad.

La relación entre escritura y programación también ayuda a cuestionar cierta cultura académica que premia la solución rápida por encima de la solución comprensible. Muchas veces el estudiante aprende que lo importante es llegar a la respuesta correcta antes que los demás, aunque el proceso quede comprimido en nombres crípticos, bloques interminables o decisiones difíciles de justificar. El examen rara vez castiga la opacidad si el resultado final es correcto. Pero la práctica profesional sí lo hace, aunque no siempre de forma inmediata. La deuda de legibilidad se acumula silenciosamente hasta que aparece como retraso, errores repetidos, dependencia excesiva de ciertas personas o miedo a tocar partes delicadas del sistema. Lo que en clase parecía una victoria, en un equipo real puede convertirse en una carga colectiva.

En este sentido, resulta valioso el hallazgo de Olivares-Rodríguez et al. (2023), quienes reportan una correlación positiva entre la producción de código limpio y las concepciones sobre la escritura en estudiantes de ingeniería en computación. Más allá del dato puntual, lo relevante es lo que sugiere: comprender la escritura como un proceso de comunicación mejora también la forma en que se programa. Esto tiene implicaciones pedagógicas claras. Tal vez enseñar programación no debería limitarse a enseñar estructuras correctas y soluciones eficientes, sino incluir también ejercicios de lectura, revisión y reescritura. Tal vez habría que preguntar no solo si el programa funciona, sino si se entiende. No solo si resuelve un problema, sino si lo expone con claridad.

También conviene notar que la comparación con la literatura no busca borrar las diferencias entre ambos campos. La programación no es literatura en sentido estricto, ni su valor depende de provocar emoción estética como lo hace una novela o un poema. El código está sometido a restricciones formales mucho más rígidas. Una coma fuera de lugar puede romper todo. La ambigüedad, que en literatura puede ser un factor expresivo, en programación suele ser un defecto. Sin embargo, esa diferencia no anula la comparación. Más bien la vuelve más interesante. Porque incluso dentro de un lenguaje altamente formal y restringido sigue existiendo espacio para la claridad, la

estructura, la intención y el estilo. Precisamente ahí radica el reto. Escribir bien cuando hay tan poco margen para el error.

La literatura también enseña algo clave sobre la forma y el fondo. Por más que a veces se separen, en la práctica no hay contenido sin forma. Una misma idea, según cómo se cuente, puede sonar increíble o puede quedar completamente confusa. En programación pasa lo mismo. Un algoritmo no existe por arte de magia fuera del código que lo escribe. La manera en que se divide una función, cómo la nombra, cómo se relaciona con el resto. Todo eso afecta si después alguien puede entenderla, modificarla o discutirla. Por eso la legibilidad no es un “detalle” que se agrega al final cuando ya se resolvió el problema. Es parte de la solución, desde el principio.

Desde esta perspectiva, resulta comprensible que algunos textos hayan querido pensar la programación también como una práctica cultural y expresiva, y no únicamente instrumental. Alonso Valero (2019), por ejemplo, sugiere que existen silimitudes más amplias entre los lenguajes de programación y la sensibilidad literaria. Incluso cuando no se trate de escribir “poesía en código”, la observación sirve para recordar que los lenguajes técnicos también organizan formas de pensar, de mirar problemas y de construir sentido. No son recipientes neutros. Elegir una forma de escribir código es, en parte, elegir una forma de ordenar el mundo dentro de un sistema.

Del mismo modo, textos de divulgación como el de Pautassi (2016) insisten en esa cercanía entre el programador y el escritor, no para confundir sus oficios, sino para subrayar un rasgo común: ambos trabajan con lenguaje, estructura y revisión. Ambos dependen de una relación exigente entre precisión y claridad. Ambos producen algo que debe sostenerse más allá del instante de su creación. Y ambos saben, o deberían saber, que escribir mal tiene efectos. En un caso, sobre la lectura. En el otro, también sobre el funcionamiento, el mantenimiento y la confianza.

Por eso, la pregunta por la relación entre programación y literatura no es una extravagancia teórica ni una ocurrencia romántica. Es una pregunta práctica sobre la naturaleza del trabajo técnico. Programar bien no es solo producir instrucciones válidas y eficientes. Es construir textos legibles dentro de un lenguaje formal. Es ordenar ideas para que otros puedan habitarlas. Es escribir con conciencia de lector. Tal vez por eso,

conforme uno madura en la disciplina, empieza a sospechar que una parte importante del oficio no consiste en dominar cada vez más comandos, frameworks o paradigmas, sino en aprender a escribir mejor dentro de ellos.

Para finalizar, no se trata de convertir a los ingenieros en poetas ni de estetizar artificialmente la programación. Se trata de reconocer una verdad: escribir bien y programar bien comparten una raíz común, que es la responsabilidad hacia quien va a leer. Esa responsabilidad exige claridad, estructura, intención y cuidado. Exige pensar no solo en lo que el programa hace, sino en cómo lo dice. Y esa exigencia, lejos de ser un lujo humanista agregado desde afuera, forma parte de lo que distingue a un buen profesional. Porque al final, tanto en la literatura como en la programación, escribir bien es una manera de respetar el tiempo y la inteligencia del otro.

Referencias

Abelson, H., Sussman, G. J., & Sussman, J. (1996). *Structure and interpretation of computer programs (2nd ed.)*. MIT Press. https://web.mit.edu/6.001/6.037/sicp.pdf

Alonso Valero, E. (2019). Aproximación a la poesía escrita en lenguajes de programación (sobre Belén García Nieto). *Signa: Revista de la Asociación Española de Semiótica*, 28, 331–349. https://doi.org/10.5944/signa.vol28.2019.25055

Knuth, D. E. (1992). *Literate Programming.* CSLI Publications. https://www-cs-faculty.stanford.edu/~knuth/lp.html

Olivares-Rodríguez, C., Valdés-León, G., Vidal-Sepúlveda, M., & Oyarzún-Yañez, R. (2023). Escritura de texto y producción de código limpio: dos realidades de un mismo proceso en estudiantes de ingeniería. *Trabalhos em Linguística Aplicada.* https://doi.org/10.1590/1983-3652.2023.41439

Pautassi, M. A. (2016). Hemingway programador. *Semana.* https://www.semana.com/impresa/articulo/literatura-y-programacion-de-computadores/54206/