Autoría y propiedad del software que construimos entre todos: ¿a quién pertenece el código?

Kaleb Coto Elizondo
Tomás André Medina Vargas

A primera vista, si vos tecleás el código, sos el dueño. Sin embargo, cualquiera que programe hoy sabe que esa noción se complica cuando se examina con detenimiento. En los proyectos de código abierto contribuyen miles de personas; las herramientas de inteligencia artificial generan bloques enteros de código a partir de una sola instrucción y la práctica habitual incluye la reutilización de soluciones publicadas en plataformas como GitHub. Encajar esta forma de trabajo colectivo en las definiciones tradicionales de autoría individual y propiedad privada requiere, cuando menos, una revisión crítica de dichos conceptos.

No es solo una pregunta teórica; tiene consecuencias concretas: quién puede usar un programa, quién puede modificarlo si contiene un error, quién puede auditarlo cuando maneja datos sensibles. Estas preguntas adquieren mayor urgencia ahora que el software está presente en la salud, la educación y los procesos electorales. Por eso vale la pena detenerse a analizar, más allá del debate abstracto sobre autoría, qué modelo de propiedad resulta más adecuado para nuestra sociedad y qué implica cada opción.

La autoría en tiempos de colaboración e inteligencia artificial

Antes, programar era una actividad individual o de equipos pequeños y cerrados. El modelo de código abierto transformó radicalmente esa imagen. Proyectos como Linux, Python o el núcleo de Android existen gracias a contribuciones distribuidas de personas en todo el mundo. Claro, Linus Torvalds inició el desarrollo del kernel en 1991, pero atribuirle la autoría completa de lo que existe hoy resulta impreciso: el sistema actual incorpora el trabajo de miles de desarrolladores que nunca tuvieron contacto directo entre sí. Como explica Paul (2015), el arte digital y el software superaron los marcos de la creación individual al introducir formas de colaboración que esos marcos simplemente no contemplaban.

La expansión de herramientas de inteligencia artificial que generan código agrega otra capa a este problema. Franganillo (2023) analiza cómo estos sistemas están modificando las bases tradicionales de la creatividad al producir contenido a partir de instrucciones textuales,

sin que exista un acto deliberado de escritura por parte del programador. Cuando vos le pedís a una IA que escriba una función completa y la integra en su proyecto, el proceso no sigue el modelo clásico de composición, aunque la decisión de usarla, adaptarla y asumir sus consecuencias sí recae sobre vos.

En ese sentido, la IA no es fundamentalmente distinta de otras herramientas que amplían la capacidad del programador sin reemplazar su juicio. Funciona de forma análoga a un compilador: transforma instrucciones en un resultado utilizable, pero a nadie se le ocurriría considerarlo coautor del programa. Si el código generado produce un error, la responsabilidad recae sobre quien lo incorporó sin revisarlo. Si funciona correctamente, el mérito corresponde a quien supo plantear el problema y evaluar la solución. Al final, escribir a mano cada línea importa bastante menos que entender qué hace ese código y por qué decidiste usarlo.

Por esa misma razón, reutilizar código publicado en plataformas como GitHub no equivale a una apropiación indebida: es parte de cómo funciona la cultura compartida del desarrollo de software. Lo que distingue a un profesional honesto no es el origen del código, sino la claridad con que reconoce las fuentes y asume la responsabilidad sobre el producto final.

Propiedad intelectual: ¿un modelo que ya no encaja?

El modelo de propiedad intelectual que regula el software fue diseñado originalmente para obras atribuibles a un creador identificable: un libro, una composición musical, una pintura. Su premisa fundamental es que existe un autor singular cuyo esfuerzo creativo merece protección jurídica durante un periodo determinado. El software contemporáneo rara vez cumple esa condición: es iterativo, se construye sobre versiones anteriores, incorpora dependencias externas y, en muchos casos, resulta del trabajo distribuido de cientos de personas a lo largo de años. Aplicar ese marco a un objeto que es por naturaleza colectivo genera tensiones que se vuelven evidentes en la práctica: las licencias de código abierto, las disputas sobre dependencias y los límites de la propiedad sobre algoritmos son problemas que el derecho de autor clásico no estaba diseñado para resolver.

El desarrollo de software opera, en gran medida, bajo una lógica de conocimiento compartido y acumulado. Los programadores adaptan soluciones que otros publicaron,

contribuyen a proyectos ajenos y construyen sus herramientas sobre código que alguien más escribió antes. Cox y McLean (2013) señalan que el software nunca es neutral: define qué operaciones son posibles y bajo qué condiciones, e impone estructuras que organizan las posibilidades de quienes lo usan. En ese sentido, decidir quién puede ver, modificar o distribuir el código no es solo una decisión económica, sino una decisión sobre quién tiene acceso y control sobre esas estructuras.

Esto no implica que el modelo propietario carezca de legitimidad. Desarrollar software requiere una inversión significativa de tiempo y recursos económicos, y es razonable que quienes lo producen busquen condiciones para recuperar esa inversión. Desde esta perspectiva, la propiedad intelectual cumple una función económica comprensible. El problema no es la existencia del modelo privado, sino la tendencia al monopolio que puede generar cuando no coexiste con alternativas reales y accesibles. Cuando no hay opción, el usuario no está eligiendo: está aceptando lo único disponible, y eso tiene implicaciones que van bastante más allá del precio del software.

Stallman (2020) define el software libre en términos de libertad, no de precio: la posibilidad de ejecutar, estudiar, modificar y redistribuir un programa convierte al usuario en alguien que comprende y controla las herramientas que usa. Esta distinción resulta especialmente relevante cuando el software en cuestión gestiona datos médicos, procesos electorales o comunicaciones privadas. En esos contextos, no poder revisar el código no es solo una limitación técnica, sino un problema de confianza institucional. Como señalaba McCullough (1996), las herramientas digitales moldean la manera en que pensamos y actuamos; si esas herramientas operan bajo control exclusivo y sin alternativas accesibles, el margen de autonomía tanto técnica como ciudadana se reduce de forma concreta.

Lo que resulta preocupante, en definitiva, no es el software privado como tal, sino la concentración de poder que puede derivarse de un ecosistema donde solo existe ese modelo. Un entorno tecnológico con diversidad de opciones —libre y privado— puede servir mejor a distintas necesidades y contextos, sin que ninguno de los dos modelos deba imponerse como el único válido.

El software como expresión: entre el código y la cultura

Hay una dimensión de esta discusión que suele quedar fuera del debate jurídico y

económico: si el software puede ser arte, la pregunta sobre a quién pertenece adquiere una capa adicional de complejidad. Knuth (1974) argumentaba que la programación no debería reducirse a la corrección técnica, sino que puede aspirar a la belleza, el estilo y la elegancia. Un algoritmo bien escrito tiene algo comparable a la claridad de un buen párrafo o a la coherencia estructural de una composición musical.

Maeda (1999) va en una dirección similar cuando reflexiona sobre cómo las herramientas digitales transforman la relación entre quien crea y lo que crea: la herramienta pasa a ser parte del resultado. Esta perspectiva complica la pregunta de la autoría, pero también la enriquece. Si el software puede entenderse como una forma de expresión cultural, entonces las decisiones sobre quién puede verlo, copiarlo o modificarlo no son solo decisiones económicas: son decisiones sobre quién tiene acceso a esa expresión y quién queda excluido de ella.

Vale reconocer que esta analogía entre software y arte tiene límites. No todo programa aspira a la elegancia estética; muchos son desarrollados bajo criterios de eficiencia y funcionalidad sin ninguna pretensión artística. Sin embargo, incluso en esos casos, el hecho de que el software se construya de forma colectiva y acumulativa hace que la pregunta sobre la autoría individual siga siendo compleja. La dimensión expresiva simplemente la hace más visible.

Esta idea tiene implicaciones profundas para la formación en ingeniería. Si el código puede ser expresión, aprender a programar no es solo aprender a resolver problemas: es aprender a comunicarse con precisión, a construir cosas que otros puedan entender y mejorar, y a participar en una conversación técnica y cultural que lleva décadas en marcha.

Una reflexión desde la ingeniería

Para quienes estudiamos ingeniería en computación en el TEC, estas preguntas no son un debate filosófico lejano. Desde el inicio de la carrera, trabajamos con sistemas basados en Linux, entornos como LibreOffice y lenguajes cuyas especificaciones son públicas. Rápidamente se hace evidente que el conocimiento técnico se acumula y no surge de la nada: aprender a programar implica apoyarse en herramientas que otras personas diseñaron, mantuvieron y pusieron a disposición. Sería contradictorio formarse en ese ambiente y luego sostener que el software es una propiedad estrictamente individual e

intransferible.

Eso no significa que la responsabilidad individual desaparezca. Hay una diferencia clara entre usar el trabajo de otros con atribución honesta y apropiarse de código ajeno sin reconocimiento. La diferencia no está en el origen del código, sino en la honestidad con que se usa. Knuth (1974) argumentaba que programar bien tiene una dimensión estética genuina, lo que implica un compromiso real con la calidad del propio trabajo. Ese compromiso no es compatible con el plagio, pero sí encaja perfectamente con aprender de la comunidad y construir sobre lo que otros han hecho.

En la ingeniería de computación, el programador se encuentra de forma permanente entre su propio esfuerzo y el trabajo colectivo: cada solución propia se apoya en lo que otros construyeron antes, y cada aporte propio puede convertirse en la base sobre la que trabaje quien venga después. Como señalaba Maeda (1999), la herramienta digital forma parte activa de la creación sin llegar a reemplazar al creador. Admitir esa interdependencia no le resta valor ni mérito al trabajo propio; simplemente lo sitúa en el contexto real en el que ocurre.

La pregunta sobre quién es el dueño del software no tiene una respuesta única, y esa dificultad no es una falla del análisis sino un reflejo de la complejidad del objeto que se estudia. El software contemporáneo es una creación acumulativa: cada programa incorpora decisiones, convenciones y código producido por quienes trabajaron antes. Los marcos legales que intentan reducirlo a la posesión de un individuo o empresa logran capturar solo una parte de esa realidad.

Tanto el modelo libre como el privado tienen lugar dentro del ecosistema tecnológico, y probablemente la respuesta más honesta no sea elegir uno y descartar el otro, sino entender en qué contextos cada uno sirve mejor y qué condiciones deben existir para que ambos puedan coexistir sin que ninguno elimine al otro. Lo que sí parece claro, retomando a Stallman (2020), es que la posibilidad de revisar el código que se ejecuta en los propios dispositivos no debería ser un privilegio técnico, sino una condición accesible para cualquier usuario.

La inteligencia artificial no suprime la autoría: la redefine. Franganillo (2023) señala que su llegada plantea preguntas sobre autoría y propiedad que todavía no tienen respuesta

consolidada. Pero si se usa con criterio, el autor sigue siendo quien decide, quien evalúa y quien asume la responsabilidad del resultado. Al final, la pregunta por el código es también una pregunta por nosotros como comunidad técnica: si el software es, como plantean Cox y McLean (2013) y como sugería Knuth (1974), una forma de expresión colectiva, entonces las decisiones sobre quién puede acceder a él y modificarlo son también decisiones culturales. Y esas decisiones merecen debatirse con la misma seriedad con que se discute cualquier otra política que afecte el acceso al conocimiento.

Referencias

Cox, G., & McLean, A. (2013). *Speaking code: Coding as aesthetic and political expression.* MIT Press.

Franganillo, J. (2023). La inteligencia artificial generativa y su impacto en la creación de contenidos mediáticos. *methaodos. Revista de Ciencias Sociales,* 11(2), m231102a10. https://doi.org/10.17502/mrcs.v11i2.710

Knuth, D. E. (1974). Computer programming as an art. *Communications of the ACM,* 17(12), 667-673. https://doi.org/10.1145/361604.361612

Maeda, J. (1999). *Design by numbers.* MIT Press.

McCullough, M. (1996). *Abstracting craft: The practiced digital hand.* MIT Press.

Paul, C. (2015). *Digital art* (3.a ed.). Thames & Hudson.

Stallman, R. M. (2020). La definición de software libre. *Communiars. Revista de Imagen, Artes y Educación Crítica y Social,* (3), 151-154. https://dx.doi.org/10.12795/Communiars.2020.i03.09