ISBN 978-950-33-1673-3

Edición de
María Paula Buteler
Ignacio Heredia
Santiago Marengo
Sofía Mondaca

Filosofía de la Ciencia
por Jóvenes Investigadores

vol.2

Filosofía de la Ciencia por Jóvenes Investigadores

vol. 2

Edición de

María Paula Buteler
Ignacio Heredia
Santiago Marengo
Sofía Mondaca

Colecciones del CIFFyH

Filosofía de la Ciencia por Jóvenes Investigadores vol. 2 / Ignacio Heredia ... [et al.]; editado por María Paula Buteler... [et al.]. - 1a ed. - Córdoba: Universidad Nacional de Córdoba. Facultad de Filosofía y Humanidades, 2022.
Libro digital, PDF

Archivo Digital: descarga y online
ISBN 978-950-33-1673-3

1. Filosofía de la Ciencia. 2. Jóvenes. I. Heredia, Ignacio. II. Buteler, María Paula, ed.
CDD 121

Publicado por
Área de Publicaciones de la Facultad de Filosofía y Humanidades - UNC
Córdoba - Argentina
1º Edición

Área de
Publicaciones

Diseño de portadas: Manuel Coll
Diagramación: María Bella
Imagen de cubierta y contracubierta: Detalle del retrato de Carpenter (1836), autora: Margaret Sarah Carpenter. Imagen de dominio público editada por Martina Schilling.
Imagen de portads interiores: Retrato de Ada Lovelace, autore desconocide, circa 1840. Seis diseños en color por Ignacio Heredia.

2022

¿Por qué es importante la metodología (de software) para la filosofía (de la ciencia)?[1]

Andrés Ilčić*

Son muchas las razones por las que creo que los filósofos de la ciencia deben prestar atención a los debates acerca de la metodología de desarrollo de software, y esto es al margen de la importancia que dicho espacio de reflexión tiene para la propia filosofía de la programación, una parte cada vez más importante de la filosofía de la ciencia de la computación.

Primero que nada, puede ser un banco de prueba interesante para cualquier teoría de la racionalidad científica. Esto puede sonar raro, especialmente ya que no toda ciencia está en el "negocio" de hacer software, pese a que el giro computacional en las ciencias sea cada vez mayor. Pero a lo que me refiero no es a la forma en la que se diseña e implementa software en las ciencias, sino más bien a las similitudes que permiten analogizar los múltiples supuestos que están detrás y que condicionan cada una de las decisiones que se toman a lo largo de los procesos característicos de estas dos prácticas, tanto en sus modos teóricos como en los experimentales.

Sin ir más lejos, podemos pensar que la relación entre las especificaciones de un programa y su implementación está tan subdeterminada como aquella relación que solemos encontrar entre la teoría y la evidencia. De hecho, bien podríamos decir que una teoría se diseña para cumplir con las especificaciones impuestas por la evidencia; mientras que su implementación, que puede tomar infinitas formas, dependerá en última instancia de múltiples factores, tanto internos como externos, que la comunidad científica en general -y un grupo o un individuo en particular- haya optado, por consciencia o tradición, como la mejor manera de proceder y satisfacer el sinnúmero de restricciones, teóricas y no tanto, provenientes del estado del conocimiento disponible en un momento dado sobre lo que se está investigando y de los objetivos que se persiguen.

En esta perspectiva, di por supuesto algo que no suele estar presente en las reflexiones más ortodoxas sobre la ciencia, ya que no suponen que

[1] Comentario a Huvelle, X. (2022) ¿Qué es un diseño Top-Down en programación? En *este volumen*. Editorial FFyH.

* CIFFyH (CONICET, UNC) / ailcic@ffyh.unc.edu.ar

la ciencia esté en el negocio de hacer y vender productos, los científicos estarían muy ocupados encontrando la mítica verdadera estructura del mundo.

Es con el surgimiento de la llamada *Big Science* que se hace patente el problema de "organizar la ciencia", problema que, en realidad, no era nada nuevo (hasta se podría decir que Tycho Brahe ya había encarado una forma moderada de Big Science), pero empresas como el Radiation Laboratory que Ernest Lawrence puso en marcha en 1931, hicieron ver que el cambio de escala requería poder articular tanto personas como instituciones que hasta entonces no se habían visto en la necesidad de interactuar tanto; mucho menos con un nuevo objetivo en común. El ciclotrón de Lawrence, junto con la experiencia que lograron los miembros de su equipo -que, entre otros grandes, incluía a Robert Oppenheimer y a Robert Wilson- probaron ser mucho más que un campo de pruebas para el Proyecto Manhattan, proyecto del que nadie podría dudar que efectivamente produjo algo; algo tan potente que no solo tenía la capacidad de destruir ciudades y cambiar el rumbo de la historia, sino también, como le gustaba decir a Popper, poner un punto final al debate sobre el realismo científico[2].

El Proyecto Manhattan mostró claramente que existía la necesidad manejar un sistema de información, creado y sostenido por agentes que realizan acciones e interactúan entre sí, pero que de alguna manera es independiente de las peculiaridades de los agentes que lo "implementan". Esto significaba que se podía dar una descripción abstracta de lo que estaba ocurriendo "ahí abajo", sin necesidad de considerar en todo momento los detalles de su constitución. Más importante aún, esto permitió pensar formas de representar y diseñar sistemas de información, tanto para actividades que ya se realizaban o para proyectos nuevos. Es por esta necesidad que surgen los lenguajes para análisis estructurado y técnicas de diseño (SADT, en inglés), facilitando el análisis de las funciones de cada módulo del sistema. La posibilidad de descomponer estas funciones en términos de funciones más primitivas que al componerlas devuelven la función original está en la base de los diseños *Top-down*, que asumen la posibilidad de una descomposición estructural-funcional, directamente

[2] Dicho sea de paso, creo que a Popper le hubiese gustado mucho la idea de Test Driven Development (TDD): diseñe primero la prueba, luego escriba el código que quiere poner a prueba.

relacionada con la metodología para trabajar con modelos para investigar "cajas negras" que la cibernética había propiciado explícitamente desde mediados de los años '40 (Rosenblueth y Wiener, 1945) y las reflexiones de Herbert Simon (1962) acerca de las analogías entre diferentes clases de sistemas complejos como los agentes inteligentes naturales y artificiales, las organizaciones y las sociedades en general. En al menos un nivel de abstracción, todos ellos pueden ser descritos en términos de procesamiento de información.

La complejidad del proyecto Apollo llevó a la NASA a adoptar esta clase de representaciones para todo lo que fuera posible, ya que permitía descomponer claramente la relación entre objetivos finales (como "llegar a la luna y volver") en términos de todos los objetivos parciales de los agentes y sus estados que estaban involucrados en la secuencia de eventos de una misión. La brillante idea de Hamilton y Zeldin fue la de sugerir ver a toda la misión Apollo como un problema de software, y en base a eso especificar los requisitos de los módulos de navegación y alunizaje y, especialmente, la interfaz entre otros agentes del sistema, tanto humanos como otros sistemas de software y hardware, siguiendo de hecho la sugerencia ya presente en el paper fundacional de Turing sobre aquello que se podía ver como computable en base a un diseño Top-down (Hamilton y Zeldin, 1976).

Aquí se puede ver con claridad por qué Huvelle (este volumen) insiste correctamente en que hay que distinguir a la programación Top-down del desarrollo Top-down. El desarrollo de un sistema sistema de software (o de un sistema visto como software), es independiente de cómo se programen sus partes; lo importante es que cumplan con las especificaciones, que han sido formuladas abstracta e independientemente de su implementación; aunque sujetas, claro, a las restricciones del conocimiento que se tiene ya del caso, de la misma forma en la que sugerí al principio que una teoría puede verse como diseñada sobre la especificación de la evidencia, sujeta al mismo tiempo al conocimiento del campo de fenómeno que se tiene en un momento determinado y los estándares que la comunidad acepte como buenos.

Creo que seguir explorando estas analogías puede ser un terreno fértil para aportar una nueva mirada a las prácticas científicas, atentos a que a medida que cambiamos la escala de observación y más elementos humanos (y otros no tanto) entran bajo la lupa, el método que se usa para

describir o planificar una práctica específica dentro del sistema general (escribir un programa o formular un modelo o un experimento) puede dejar de funcionar y la metodología de análisis debe volverse entonces tanto más compleja, si es que no, quizás, imposible de describirse Top-down a medida de que más y más elementos entran en nuestra descripción del proceso y no pueda ya éste verse como software; y mucho menos susceptible de una descomposición funcional, dada la cantidad de ciclos de retroalimentación presentes y las incertidumbres propias de nuestra limitada capacidad epistémica.

Referencias Bibliográficas

Hamilton, M., y Zeldin, S. (1976). Higher order software: a methodology for defining software. *IEEE Transactions on Software Engineering, SE-2(1)*, 9–32. En: https://doi.org/10.1109/TSE.1976.233798.

Rosenblueth, A., y Wiener, N. (1945). The role of models in science. *Philosophy of Science, 12(4)*, 316–321. En: https://doi.org/10.1086/286874.

Simon, H. (1962). The architecture of complexity. *Proceedings of the American Philosophical Society, 106*, 467–482.