Tabla de Contenidos

Episodio 35: El software, un parque de abstracciones

Hoy le daremos vueltas a una palabra que asociamos con la tecnología pero sobre la que se puede arrojar una mirada filosófica: el software, tratando de responder a la preguna ¿qué es el software?. Este es un concepto que se suele contraponer con otro denominado “hardware”, creándose así la dualidad software/hardware que se une a otras famosas dualidades estudiadas por la filosofía, tales como mente/cuerpo o la dualidad onda/partícula en el campo de la física. ¿Puede ayudarnos a entender estas otras dualidades la comprensión de en qué consiste la dualidad software/hardware?

“La filosofía no sirve para nada” es un podcast sin pretensiones en el que reflexionaremos sobre el presente.

Participan: José Carlos García @quobit, Juan Antonio Torrero @jatorrero, Sergio Muñoz @smunozroncero, Joaquín Herrero @joakinen, Juan Carlos Barajas @SociologiaDiver

Fecha	2 de noviembre de 2020
Participan	José Carlos García @quobit Juan Antonio Torrero @jatorrero Sergio Muñoz @smunozroncero Juan Carlos Barajas @SociologiaDiver Joaquín Herrero @joakinen
Descarga	Puedes descargar todos los episodios en iVoox, en Spotify, en iTunes, Google Podcasts y en nuestro canal de Telegram. Si tienes un lector de podcasts que admite enlaces RSS, este es el enlace RSS a nuestro podcast.
Sintonía	Mass Invasion, Dilo, álbum **Robots** (2004)
Fotos	Whats that? (5), Steve Jurvetson, Flickr
Intro	Conversación con Alexa
Twitter	En @FilosofiaNada publicamos noticias que nos interesan y conversamos.
Canal Telegram	Puedes seguir la preparación de nuevos episodios suscribiéndote al canal @FilosofiaNada en Telegram
Grupo de opinión	Únete a nuestro grupo de opinión Opina FilosofiaNada para opinar sobre el episodio en preparación y enviarnos audios con preguntas o críticas con humor para nuestra intro

Definiciones

Software

Wikipedia en español:

- Se conoce como software al soporte lógico de un sistema informático, que comprende el conjunto de los **componentes lógicos** necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos que son llamados hardware. La interacción entre el software y el hardware hace operativo un ordenador (u otro dispositivo), es decir, el software envía instrucciones que el hardware ejecuta, haciendo posible su funcionamiento.

Wikipedia en inglés:

- Computer software, or simply software, is a collection of data or **computer instructions** that tell the computer how to work. This is in contrast to physical hardware, from which the system is built and actually performs the work. In computer science and software engineering, computer software is all information processed by computer systems, programs and data. **At the lowest programming level,[clarification needed] executable code consists of machine language instructions supported by an individual processor—typically a central processing unit (CPU) or a graphics processing unit (GPU). A machine language consists of groups of binary values signifying processor instructions that change the state of the computer from its preceding state.**

Oxford Languages:

- Conjunto de **programas** y rutinas que permiten a la computadora realizar determinadas tareas. Origen: Préstamo del inglés software, compuesto por soft ‘blando’ y ware ‘utensilios, objetos’. Término creado por los ingenieros informáticos americanos por analogía a hardware.

Diccionario RAE:

- Conjunto de **programas, instrucciones y reglas informáticas** para ejecutar ciertas tareas en una computadora.

Hardware

Wikipedia Español:

- La palabra hardware en informática se refiere a las partes físicas, tangibles, de un sistema informático, sus componentes eléctricos, electrónicos, electromecánicos y mecánicos.

Wikipedia Inglés:

- Computer hardware includes the physical parts of a computer, such as the case,[1] central processing unit (CPU), monitor, mouse, keyboard, computer data storage, graphics card, sound card, speakers and motherboard. **By contrast, software is the set of instructions that can be stored and run by hardware.**

Oxford Languages:

- Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático. Origen: Préstamo del inglés hardware, compuesto por hard ‘duro’ y ware, elemento que entra en la formación de muchos términos ingleses con el valor de ‘utensilio, artículos’. El término fue creado en los años cincuenta por ingenieros informáticos junto a software (V.).

Diccionario RAE:

- (Informática) equipo (‖ conjunto de aparatos de una computadora).

CUESTIÓN: ¿LOS ESTADOS ELECTRÓNICOS SON HARDWARE O SOFTWARE? (Esta cuestión es paralela a la de los “estados mentales” en la cuestión mente/cuerpo)

Estatuto filosófico del software

Basado en el capítulo 12 (“Software, Abstraction and Ontology”) del libro Philosophy and Computer Science, Timothy Colburn, 2000, Taylor & Francis.

Book Review: A Philosophy of Software Design | Johz Blog
https://johz.bearblog.dev/book-review-philosophy-software-design/

INTRODUCCIÓN DEL CAPÍTULO

Traditional philosophical problems in ontology, that is, problems concern- ing what kinds of things there are, and how to classify certain individual things, rarely hold more than academic interest. The outcomes of de- bates over the ontological status of bare particulars, for example, will not make headline news or affect the lives of nonphilosophers very much. Of course, a decision to make an ontological commitment often does affect how we do science, as when the conjecture that more than 90 percent of the universe is composed of unobserved “dark” matter prompts lengthy and risky experiments to capture a fleeting glimpse of a neutrino. Still, although the outcomes of these experiments hold enormous implications for cosmology, they will not affect the day-to-day lives of ordinary people whose work involves business, finance, trade, or technology.

However, the ontological status of computer software has emerged as an issue that may hold interest both for academic philosophers and for those involved in the formation of public policy for the international marketplace. As reported in a recent issue of Scientific American} the apparent dual nature of software has sparked a debate involving free speech and the applicability of the International Traffic in Arms Regu- lations. A related controversy concerning policy governing intellectual property rights also appears to have its roots in a perceived dual nature of software.

The aim of this chapter is not to propose solutions to these debates, but to bring to light and analyze the philosophical assumptions under- lying them. I will describe the prima facie reasons for upholding soft- ware’s duality, along with the ontological implications. Along the way, I will expose a misconception, namely, that software is a machine made out of text. I will also show how this view is related to the view, already considered in chapter 9, that computer science is merely a branch of pure mathematics. By describing the actual practice and infrastructure surrounding software creation today, I will support a view of software as a concrete abstraction, and try to explain what this could mean from an ontological point of view.

La doble naturaleza del software

The medium of an algorithm's description might be a natural language like English, if it is being described informally, or it might be an artificial language like C++ or Java. In the latter case, the algorithm is encoded in the text of a computer programming language.

The controversy reported in Scientific American concerns an author of a book on applied cryptography who was prohibited by the State Department from exporting the book because it included as an appendix a floppy disk containing program text expressing algorithms for encryption software. […] However, were the floppy disk not included with the book, the book would have been freely exportable even though the program text on the floppy disk was also printed in the book. **As text on the printed page, the software was sim- ply a product of every American's right to free speech. But as magnetic representations of the same software on a different medium, it qualified as part of a potentially dangerous mechanism and a threat to national security.**"

There is a philosophically interesting aspect of this controversy, in that the two sides naturally appeal to ontological characterizations of software that appear to be inconsistent: **the libertarian side wants to see software as text and therefore freely distributable, while the regulatory side wants to see software as mechanism and therefore subject to export restrictions.**

The text/mechanism dichotomy is also at the root of controversies over software copyrighting.

In A New View of Intellectual Property and Software the authors point out that laws implementing copyright and patent policy were written at a time when copyright was considered to apply only to literary work, while patents were considered to apply only to the artifacts of technology, and these were mutually exclusive domains. **The important assertion here is that "software is a machine whose medium of construction happens to be text**."

From a philosophical point of view the assertion raises two obvious questions:

1. How can a machine be constructed out of text?
2. How can software, which is constructed using statements written in a formal programming language, be considered a machine'?

When we distinguish between physical and virtual machines, we see that the first question is not puzzling.

The nature of software's duality can be illuminated by understanding the distinction between software's **medium of description** (formal language) and its **medium of execution** (circuits and semiconducting elements).

Strictly speaking, when you look at printed program text […] what you "see" is an **abstraction**. The software itself, which causes the machine to execute the computational process, is constructed not out of text, but out of electron charges, magnetic fields, pulses of light, and so on, which are not abstractions, but to which we give an interpretation as binary numbers.

The idea that software is a machine built from text gives rise to the notion that **software is some kind of mysterious entity which is abstract and which possesses causal power.**

Nota: El vendedor de un NewBrain en una tienda especializada se sorprendía de la capacidad de encender y apagar una pantalla incorporada en el ordenador con tan solo una instrucción de software: OPEN #0,3

Cuando surge la cuestión filosófica entre si el software es un sistemas abstracto o uno causal la respuesta no debería ser combinar las categorías de texto y mecanismo en una categoría aún más desconcertante que admita ambos. Program Verification: The Very Idea., Fetzer

The incalculable power and utility of tools like compilers, assemblers, and link-loaders masks the fundamental fact that **to program a machine is to put its memory into a state which when presented as input to circuits will produce more state of a desired kind**. The various levels of formal language that we use today to describe these states are abstract tools for achieving this.

The remarkable thing about software is not that it is a machine whose medium of construction is text, but that a textual description of a piece of software is involved in its own creation as states of memory elements. The development over time of tools many programmers take for granted is the reason we can do this relatively easily.

Formal languages are indispensable abstractions for thinking about software, but there is no reason to think that software is a machine made of text.

Software como una abstracción concreta

The particle and wave characterizations of light were both physical characterizations (albeit ones which nineteenth-century classical mechanics had come to place “in logical opposition to each other.”)

By contrast, the duality inherent in a “concrete abstraction” crosses metaphysical categories, and deserves at least as serious philosophical scrutiny as the Copenhagen interpretation. While computer scientists often enthusiastically embrace this duality, a metaphysician will view it as a puzzle to be explained. **How can something, namely a computer program, be at once concrete and abstract?**

Prólogo del libro **Concrete Abstractions: An Introduction to Computer Science**:

A primera vista, el título de este libro (Abstracciones concretas: Introducción a la informática) es un oxímoron. Después de todo, el término abstracción se refiere a una idea o descripción general,

divorciada de los objetos físicos reales. Por otro lado, algo es concreto cuando se trata de un objeto en particular, tal vez algo que puedas manipular con las manos y mirar con los ojos. Sin embargo, a menudo se trata de abstracciones concretas. Considere, por ejemplo, un procesador de textos. Cuando usa un procesador de texto, probablemente piense que realmente ha ingresado un documento en la computadora y que la computadora es una máquina que manipula físicamente las palabras en el documento. Pero en realidad, cuando "ingresa" el documento, no hay nada nuevo dentro de la computadora; solo hay diferentes patrones de actividad de cargas eléctricas que rebotan de un lado a otro entre los capacitores. Incluso el programa al que usted llama "procesador de texto" es una abstracción: es la forma en que los humanos elegimos hablar sobre lo que, en realidad, son más cargas eléctricas. Sin embargo, al mismo tiempo que abstracciones como los "procesadores de texto" y los "documentos" son simplemente formas más convenientes de describir patrones de actividad eléctrica, también son cosas que podemos comprar, vender, copiar y usar.

El estatus ontológico de las abstracciones concretas

Think of the methodological framework employed by philosophers to approach the mind/body problem: the taxonomy of solutions in terms of monism and dualism provides a useful metaphor, if not an entirely appropriate language, for describing programs as concrete abstractions.

Monismo

Afirma que la dualidad es falsa, que el objeto solo es A o B pero no ambas cosas

Ejemplos de monismo son el materialismo y el idealismo.

Variedades de monismo:

1. Teoría del doble aspecto (afirma que la dualidad es solo aparente, el objeto se puede ver desde el aspecto A o desde el aspecto B pero el objeto no es ni A ni B)
2. Reduccionismo (uno de los aspectos es verdadero y otro falso: el objeto es A, y desde A se puede explicar que se pueda ver como B)

Philosophical implications of quantum mechanics, Norwood Russell Hanson

Dualismo

Afirma que tanto A como B son sustancias físicas y existen realmente. Hay objetos que son/pertenecen a A y B

Variedades de dualismo:

1. Interaccionismo dualista: hay una relación causal entre A y B (entre lo físico y lo mental, por ejemplo)
2. Epifenomenalismo: la causalidad entre A y B va en una única dirección
3. Armonía preestablecida: A y B son mundos separados e independientes y las correlaciones que se observan entre A y B no pueden ser casualidad (que también es una opción) sino que hay en el diseño del universo establecida una correlación entre ambos mundos. El mundo está hecho así. Esa armonía preestablecida entre A y B es establecida por… Dios.

Armonía pre-establecida

Esta última, sin embargo, es la que mejor explica el estatuto del software como un texto escrito en un lenguaje formal (software) y que tiene efectos causales en los impulsos eléctricos que suceden en otro mundo (hardware) debido a una correlación explícitamente diseñada por ingenieros y programadores. Al compilar y linkar un programa se hace ese paso entre dos mundos ontológicamente distintos: en el programa solo hay letras y números y en el hardware solo hay impulsos eléctricos pero ambos mundos han sido diseñados exprofeso para correlar entre ellos y se ha fabricado una causalidad que puede ir del software al hardware (ingeniería) o del hardware al software (ingeniería inversa)

Lo que demuestra esto es que comprender la dualidad hardware/software no nos sirve para comprender las dualidades de la naturaleza porque la dualidad hardware/software la hemos fabricado nosotros y hemos hecho las funciones de un “dios” en ese universo.

Al final Colburn hace una reflexión sobre que ninguna idea filosófica se queda antigua. Son armas conceptuales que pueden servir para explicar el presente que tenemos o quizás sirvan para explicar otros presentes que vendrán, pero nunca se debe descartar ninguna idea. Un ejemplo de esto es el concepto de “armonía preestablecida”.

Referencias


```
1. Wallich, “Cracking the U.S. Code.”
2. Cormen, Leiserson, and Rivest, Introduction to Algorithms, 831-836.
3. Davis, Samuelson, Kapor, and Reichman, “A New View of Intellectual
Property and Software,” 23.
4. Fetzer, “Program Verification: The Very Idea.”
5. Hailperin, Kaiser, and Knight, Concrete Abstractions: An Intro-
duction to Computer Science. I
6. Reichenbach, “The Principle of Anomaly in Quantum Mechanics.”
7. Hanson, “Philosophical Implications of Quantum Mechanics,” 42.
8. Shaffer, “Mind-Body Problem.”
```


Otras referencias

- Layers of Abstraction in Computing (vídeo)
- Múltiples referencias de “otro nivel de abstracción” en el curso acelerado de Informática:
  - Computer Science Crash Course playlist: https://www.youtube.com/playlist?list=PL8dPuuaLjXtNlUrzyH5r6jN9ulIgZBpdo
  - Registros y RAM: https://youtu.be/fpnE6UAfbtU?t=496
  - Instrucciones y programas: https://youtu.be/zltgXvg6r3k?t=453
  - Los primeros lenguajes de programación: https://youtu.be/RU1u-js7db8?t=232