DOI: 10.25100/pfilosofica.v0i51.10114

AN ONTOLOGY OF SOFTWARE: SERIES, STRUCTURE AND FUNCTION[1]

Jorge Francisco Maldonado Serrano
Dairon Alfonso Rodríguez Ramírez
Paul B. Caceres
Johann Farith Petit Suárez
Universidad Industrial de Santander, Bucaramanga, Colombia.

Abstract

This article proposes a guideline to develop an ontology of software. The first section gives a brief introduction to the importance of such ontology as a possible conceptual grounding for the philosophy of software, philosophy of computing and philosophy of information. The second section presents the background of the scope of this article in terms of both a symbolic and materialistic approach to software. The third section deploys the basic guidelines with the expositions of the two dimensions of software: the serial dimension and the structural dimension. The first dimension consists of three series, while the second in the exposition of the structure of any program. The fourth and last section will deal with a better understanding of what we can call the digital universe.

Keywords: *Ontology; Philosophy of Software; Series; Digital Universe; Technology.*

[1] This article is the result of two research projects sponsored by the Industrial University of Santander (1764) "Ethnography of the Subjectivities of the Digital Thought" & (2403) "Experimental Moral Philosophy for the development of Citizen Competences"

Cómo citar este artículo: Maldonado-Serrano, J. F., Rodríguez-Ramírez, D. A., Cáceres B. P., & Petit-Suárez, J. F. (2020). An ontology of software: series, structure and function. *Praxis Filosófica*, (51), 115-132. doi: 10.25100/pfilosofica.v0i51.10114

Recibido: 29 de abril de 2020. Aprobado: 3 de junio de 2020.

ISSN (I): 0120-4688 / ISSN (D): 2389-9387

Una ontología de software: series, estructura y función

Jorge Francisco Maldonado Serrano[2]
Dairon Alfonso Rodríguez Ramírez[3]
Paul B. Caceres[4]
Johann Farith Petit Suárez[5]

[2] Doctor en Filosofía de la Universidad Autónoma de Madrid. Mágister en Filosofía de la Pontificia Universidad Javeriana de Bogotá. Licenciado en Filosofía y Letras de la Universidad Santo Tomás de Bogotá. Profesor titular de la Escuela de Filosofía en la Universidad Industrial de Santander. Investigador del Grupo Tiempo Cero.

ORCID: 0000-0002-7707-154X **E-mail:** jmaldona@uis.edu.co

[3] Doctor en humanidades (filosofía) por la Universidad Autónoma Metropolitana de México. Mágister en Ciencias Cognitivas de la Universidad de Morelos, México. Filósofo de la Universidad Industrial de Santander. Investigador invitado en el Instituto Max Planck de Antropologia Evolutiva y Profesor Auxiliar de la Universidad Industrial de Santander. Director del Grupo Tiempo Cero.

ORCID: 0000-0002-5183-7121 **E-mail:** darodri@uis.edu.co

[4] Magister en Filosofía de la Universidad Industria de Santander, Abogado y Filósofo. Joven investigador de Colciencias (2019-2020) vinculado al grupo de investigación Politeia de la UIS y Teoría del Derecho y formación jurídica de la UNAB. Profesor de cátedra de la Escuela de Filosofía de la Universidad Industrial de Santander, UIS y del programa de Derecho de la Facultad de Ciencias Jurídicas y Políticas de la Universidad Autónoma de Bucaramanga, Unab.

ORCID: 0000-0002-4561-751X **E-mail:** pcaceres@unab.edu.co

[5] Doctor en ingeniería Eléctrica de la Universidad Carlos III de Madrid (UC3M), Mágister en Ingeniería eléctrica e Ingeniero Electricista de la Universidad Industrial de Santander (UIS), Bucaramanga, Colombia. Actualmente se desempeña como Decano de la Facultad de Ingenierías en la Universidad Industrial de Santander (UIS-Colombia) de donde es profesor desde 2001. Sus áreas de investigación incluyen calidad energética, potencia electrónica, redes inteligentes, sistemas de energía eléctrica y enseñanza de la ingeniería.

ORCID: 0000-0003-2283-3268 **E-mail:** jfpetit@uis.edu.co

Resumen

Este artículo propone dar unos lineamientos para desarrollar una ontología del software. La primera sección da una breve introducción a la importancia de tal ontología, entendida como una fundamentación conceptual para una filosofía del software, una filosofía de la computación y una filosofía de la información. La segunda sección presenta el trasfondo del enfoque de este artículo en términos de una posición materialista y una posición simbólica. En la tercera sección se despliegan los lineamientos básicos de dicha ontología con la exposición de las dos dimensiones del software: la dimensión serial y la dimensión estructural; la primera que consiste en tres series y la segunda que consiste en la exposición de la estructura lógico formal de cualquier programa de computación actual. La cuarta y última sección da cuenta de las posibles ganancias que se pueden obtener gracias a haber asumido este enfoque ontológico del software, lo cual permite tener más claridad a la hora de hablar del universo digital.

Palabras clave: *ontología; filosofía del software, series, universo digital.*

AN ONTOLOGY OF SOFTWARE: SERIES, STRUCTURE AND FUNCTION

Jorge Francisco Maldonado Serrano
Dairon Alfonso Rodríguez Ramírez
Paul B. Caceres
Johann Farith Petit Suárez
Universidad Industrial de Santander, Bucaramanga, Colombia.

I. Ontology and Ontology of Software

Ontology, the systematic study of being *qua* being, has addressed objects in their different modes of existence, their kinds and structures. As such, ontology traditionally puts forward that existent objects must be clearly distinguished from subsistent or ideal objects. These objects clearly do not exist as physical objects, but rather as entities whose being is both non-temporal and non-spatial[6]. In any case, traditional ontology assumes that it is possible to give an abstract description (ontological) of the modes of existence of particular types of objects. In fact, distinguishing among different categories of objects is already the main task of ontology, as differences among them are grounded in their particular modes of existence.

Martin Heidegger, a German philosopher who deeply influenced various areas of thinking (not only philosophy), developed a pioneering discourse on technics and technology which most of the theories on philosophy of technology still discuss. Paradoxically, Heidegger's original contribution to philosophy was not in philosophy of technology, for he thought such classification was an outrage to philosophy, but to the area of ontology. He proposed the idea of existential ontology as the authentic task of philosophy. He understood that any ontological inquiry must start with the exposition of the mode of existence of the 'who' that asks for the being and, at the same time, of the 'who' that can answer such a question. According to Heidegger, in both cases the 'who' is no other than the existence of the human being (Heidegger, 2010). This does not mean that the existence of objects depends on human subjectivity, but rather that any determination or explanation of

[6] We do not propose to think of ideal objects as eternal, but rather as objects that do not decay in time. In this sense we can understand that these objects have a starting moment for us humans, but after that moment they would not deteriorate or decline. One could argue that there is a mental space, but the non-spatiality we argue refers only to space in a physical sense, leaving other kinds of spatialization out of this discussion.

how something exists is essentially conditioned by the possibilities offered to human existence to grasp objects' existence. Following Heidegger's lead, we will assume that the mode of existence of software is related to human existence. Nevertheless, in this paper we will not directly consider the relation of human existence to software's existence, but only the previous analysis that will prepare us for an upcoming exposition of such relationship, crucial to understanding software.

An ontological approach within the philosophy of technology has been regarded with a certain degree of reserve. In this sense, contemporary philosopher Andrew Feenberg has an emblematic stance: "Ontological holism is of course an interesting notion, but the critique of technological rationality does not require it. A non-ontological formulation of a critical theory of technology is possible on terms that leave natural science out of account." (Feenberg, 2002, p. 175). The problem for us is not whether ontology is a necessary speculation or a senseless word-game, but rather a way to address the question of the nature of software, so to speak, a problem that supposes an accurate understanding of software in its technical relations to humans (users and programmers). An ontology of software[7], in this sense, would be a theoretical step towards a theory of the digital universe as a theory that can encompass the actual situation of our technological society[8].

II. Approaches to Software

The guidelines we would like to consider in section 3 will be better understood against the background of prevalent philosophical[9] visions about

[7] Yuk-Hui's has a different approach to software in his ontological theory of digital objects. His key category is *Data* (Hui, 2016). From our perspective, computer data has its condition of possibility given by a software. Data for a computer is something existent only within a frame of an algorithm established by a software that does the task of diving a precise codification to data. Anyhow, the direction of Yuk-Hui's analysis and ours seem to converge in that we do need an ontology of the digital and that we must understand the basic condition of possibility of any digital object. We argue software is such a condition, even before data.

[8] Álvaro Monterroza (2018) has developed an ontological analysis of the heterogeneous nature of technical artifacts which we can consider an higher level analysis in which our ontological analysis fits, even though he does not properly discusses digital objects.

[9] It is very difficult to establish a clear limit between the philosophical tradition that discusses this problem and a more technological tradition. Timothy R. Colburn (1999) is a very well-known author in the technological field for discussing the problem of the dual nature of software or its ontology, just as David R. Koepsell (2003)like an invented machine or process, or an original expression to be copyrighted, like drawings and books? This distinction is artificial, argues Koepsell, and is responsible for the growing legal problems related to intellectual property law. Computer-mediated objects are no different from books, songs, or machines and do not require any special treatment by the law. The author suggests revisions to the legal framework itself which prevent this artificial and problematic

software. In the present section, we will discuss critically two of them: a materialistic understanding of software and a symbolic interpretation. Although there are many possible authors to consider, we will study the works of David Berry and Luciano Floridi as their respective thoughts have become very influential in recent debates about software[10].

David Berry advocates for understanding software in its material aspect. He understands code and software as two sides of the same coin (Berry, 2011). For him, code is the material side and software a more symbolic reality:

> Software is therefore 'not only "code" but a symbolic form of writing involving cultural practices of its employment and appropriation' (Fuller 2008: 173). [...] we can think of code as the 'internal' form and software as the 'external' form of applications and software systems. Or to put it slightly different, code implies a close reading of technical systems and software implies a form of distant reading. [...] Perhaps the most important point of this distinction is to note that code and software are two sides of the same coin, code is the static textual form of software, and software is the processual operating form. (2011, p. 32).

Code is, for Berry, the text in a programming language that determines software. Therefore, code, as a text, can be printed or written, saved on a media or even memorized in someone's brain, and it is this aspect of code that probes its materiality. Although we consider this an interesting distinction, we claim that software as such is a better basis for our analysis. Assuming software as an opening category we will be in a better position to understand the impact digital technology has in our modern world and interpret code,

distinction, and simplifies the protection of all intellectual property.","ISBN":"978-0-8126-9537-3","language":"en","number-of-pages":"164","publisher":"Open Court Publishing","source":"Google Books","title":"The Ontology of Cyberspace: Philosophy, Law, and the Future of Intellectual Property","title-short":"The Ontology of Cyberspace","author":[{"family":"Koepsell","given":"David R."}],"issued":{"date-parts":[["2003",2]]}},"suppress-author":true}],"schema":"https://github.com/citation-style-language/schema/raw/master/csl-citation.json"} , for discussing the nature of the so-called cyberspace. This paper will not discuss such a trend of discussion, although it may seem that it should. Our only argument in favor of such a decision is to offer a perspective from which to offer a detailed analysis in a future research.

[10] Stiegler (1994; 1996; 2001) does not develop an ontology of software, of information or of the digital, but we could consider his analysis of technics deserves special consideration as a predecessor to this proposal if the aim of this paper were not be limited to the technicity of the digital software.

not as a mere material reality of software, but a series where the symbolic (programming language) is already present.

Through code, Berry characterizes software as a material reality. Assuming Latour's perspective, he understands the process of software production. This process ranges from the code design to licensed software, and can be described as a material process. Here, we find a mistake which consists in assuming a process as a material reality. By reducing software to its effects or material realizations and to its dependence on material circuits, Berry does not properly account for the symbolic reality software purports. The software's logico-mathematical structure of the machine language and the symbolic images it can display, as well as the hardware processes, require more than a material conception of software.

We suspect that one reason for Berry's point of view would be the implicit acceptance of the standard distinction software/hardware. But this would not be helpful either, for it supposes a hiatus, an ontological break. A dualistic approach cannot give an account of how hardware relates to software, for the characteristic of any dualism is precisely to leave both sides without a connection. But we, empirically, know just the contrary, that there is some kind of continuity between software and hardware.

Berry's thesis primarily aims to macro-processes such as political, economic and law processes, which are essential for his understanding of code: "Getting at the materiality of code has to take into account its physicality and obduracy, but also the 'code work' and 'software work' that goes into making and maintaining the code (e.g. documentation, tests, installers, etc.), the networks and relationships, and the work that goes into the final shipping product or service." (Berry, 2011, p. 32). We consider these aspects very worthy for any ontology of software, but these must be completed with an additional distinction. Our proposal is to understand software as a complex of two dimensions, the serial dimension and the structural dimension. The serial dimension would conjugate symbols, matter and energy, while the structural dimension conjugates code, intentionality and machine language. As we have said, this article will only present the first dimension. From this point of view, the material conception and the hardware/software distinction can be understood as an illusion, based on a hylemorphic conception.

The distinction between hardware and software is a conceptual distinction based on the perception that a computer is a compound, a material and formal part. The material part is composed of circuits, while the formal part consists of software. This recalls the cartesian dualism of *res extensa* and *res cogitans*. But this way of perceiving the computer and, thus, software does not do justice to its complex reality.

On the one hand, software is understood as something that can be separable from hardware as if it could exist independently from its physical realization. But we all know that software can only function if it is coded in and for a particular type of hardware structure. The fact that we find similar programs on different hardware platforms may reinforce this dualistic perception. In such cases, what is shown on the screen seems alike (if not the same). So, according to the dualistic perception, the codification behind the screen is not a particular codification for a particular type of hardware, but a generic code that can be implemented by any kind of hardware. What we want to stress is quite the opposite, that it is not correct to assume that codification is independent of the hardware for which it is intended. This nuance is not considered under a dualistic view.

On the other hand, this perception supposes the ontological independence of hardware, otherwise, it would not make sense to distinguish it from software. But, just as mentioned above, computer hardware is not a stand-alone machine, i.e., any system of circuits needs a program for it to function properly as a computational device. Hardware alone would be nothing more than a bunch of plastic and metals. Rightly acknowledged hardware without software would not be hardware at all. One could argue that some of the actual computer devices work as stand-alone machines (LED T.V.s, washing machines, dishwashers, Blu-ray players and the like), but all of these machines work only under the condition they have some software embedded in the hardware.

Once we have seen the obstacles and difficulties for an ontology in which software is considered just material, now we can examine a more symbolic approach to software exemplified by Floridi's informational ontology.

Floridi (2011) advocates for an informational ontology, in opposition to a digital ontology. He makes his case explicit as to what digital ontology he detaches from: "digital ontology, according to which the ultimate nature of reality is digital, and the universe is a computational system equivalent to a Turing machine, should be carefully distinguished from informational ontology, according to which the ultimate nature of reality is structural, in order to abandon the former and retain only the latter as a promising line of research." (2011, p. 36).

The problem for us is that his claim about information is applied to different kinds of phenomena which should not be homologated as if information were a unified phenomenon. Even if he rightly works out the manifold nature of information, all its meanings and different contextual uses (Floridi, 2010), we do not pretend to discuss this aspect of his research within the scope of our proposal. Suffice is to say that before admitting information as the basic ontological dimension, we would rather look at technical mechanisms that produce it. Once we can grasp this production an understanding of the differences, similarities or unity of information can be established. But this implies extensive research, to assess if such generalization under the concept of information gives an acute account of the phenomena.

On the contrary, software seems to be a very precise reality of which we can give a precise account without being forced to extend its genesis to Charles Babbage's Analytical Engine. The interesting aspect of software, the only digital object as it were, is that it has a very complex way of existence. Restricting it to the idea of digital information, i.e., information computed by digital computers, is to forget we always need a previous software that would capture information as data, compute it and produce new data. Even if we were to consider the code of software as information, we need to notice that previous running software is what enables the production of new computer code that would work as new software. Put it in other words, any digitized information or computer data needs a pre-existing structuring software that gives information (and data) its structural possibility of existence. In this sense, software is ontological prior to information.

Thus, an ontology of software, in the sense we are talking, does not reduce software to a material reality nor to an abstract informational realm. Let us present the two basic guidelines to be considered in an ontological analysis of software.

III. Ontology of Software: Series and Structure

The main problem with an ontology of software is that software is not a thing or an entity as we are used to thinking about them. Software, undoubtedly as an expression, refers to the kind of *thing* that can only exist or is accepted as existent if various processes and various components function. But this functioning is taken for granted, for it is quite complex, and it must be obfuscated if the expression 'software' is intended to be used as if it were referring to an object or a thing. The object or thing we think software refers to does not really exist as such, we argue.

The first task for an ontology of software is to think about the complexity of software. Therefore, we consider this a first approach which can and has to be enriched, criticized and perfected. Let us examine what we consider as the first ontological dimension of software.

3.1 Serial dimension of software

Consider three series: energy flux, circuitry and code. The first series, the energy flux, can be understood just by considering the fact that any given digital computer consumes energy with a standard voltage supply (±100V through ±240V, depending on the country). Electrical energy is and has to be flowing through the circuitry in order for us to say there is software running. But the important aspect to highlight is that the input voltage is regulated according to the circuit that needs it (screen, flash drive, CPU, cooling system, and the like). Each circuit consumes energy at some specific voltage level and at a proper frequency. Let us not pass into oblivion that network connections, either wired or wireless, are also the flows of energy necessary for a computer device to work properly. The computer transduces these fluxes permanently in order to achieve electronic homogeneity and stability. Therefore, our first claim is that the computer organizes electrical fluids. This electrical organization is made in accordance with the logic that is embedded in the circuits. This logic is fit to the mathematical theory of information and, based on mathematical theory of probability, is at the base of the organization of signals of energy that the computer receives and sends. In this context, the energy flux is in-formed or, more precisely pre-formed, as it is regularized, or it is given a special voltage. We can say that a series of energy fluxes is part of the computational process we think of as software.

The second series, the circuitry, can be understood as space or a place where and through the electrical energy is properly driven. Circuits are organized on a plastic board, on a physical series, that strictly speaking is material. This series is organized in response to the physical properties of the materials used (copper, silicon, gold, tin, quartz). Each of the circuits is constructed with specific materials that give them their computational capabilities, so to speak. This means that the computational possibilities hardware has are in direct logico-mathematical relation to the materials it is made of. Thus, the series of circuitry is configured logically so that it can stand the energy flow and pre-form it. Let us notice, therefore, that both series converge on each other for energy flows, as they are pre-formed, through the materials which are configured in order to achieve such a flow.

The third series corresponds to the different levels of codification or programming we find in computers. From the hard-wired firmware to the drawing of figures on design software, we can detach various levels of computer language if we look at the programming and the functioning.

The Programming aspect of the code series can be disaggregated in five levels. Computer machine programming includes a very low-level code that is hardwired in every one of its components in order for them to work. It is a code embedded in hardware, which we hardly acknowledged as software in our normal way of understanding software. This hardware code will be the last level of code to be executed in order to run a higher-level code. We can also differentiate the code in machine language that runs through the CPU from the one that runs in the mainboard. But, for the CPU to run any machine-language program, it must be prepared by the operative system (OS), a program that enables the circuitry to respond to the flow of commands.

Let us not forget that the OS and the programs executed in any OS need to be translated into machine language. This means the compiler program plays a very important role in the art of programming because it is responsible for taking any code programmed by a human being and turning it into the code that can flow through the CPU and through the rest of the hardware.

Finally, we have the programs in computer languages that, likewise, are particular machine language code that presents a humanly understandable code to test the ideas of the programmer. These computer languages are considered high-level languages inasmuch as they are very distant to computer language and closer to the human way of speaking. In any case, still no computer language is near a normal language, for it is very simple and logical. Normal language is full of fuzzy meanings and uses, imprecise or variable constructions, and the like. Computer language, although can be structured differently for equal ends, has to be very precise in order to avoid errors. Strictly speaking meaning is not an essential part of computer language as it is to normal human language, it is pure syntaxis. These five levels are not the whole of the code series, for in functioning the picture is quite different.

The functioning code, in the sense that a code must be run in a computer for it to work can easily be distinguished from code as a list of lines of programming. Running code only has two levels: machine language, in which everything (that is presented on screen, for example, or in any other output device or, in general, in any circuit) really happens; and hardwired functioning code. If we were to look with our eyes, as in the movie *Tron*, inside the hardware and try to perceive what is happening inside the wires, we would only perceive flows of energy. The interesting point is that this flow is organized logically in machine language. A running code inside the computer is a complex flow of electricity that can be represented as simple zeroes and ones. This means that, for example, the running code that makes the mouse pointer a visual reality on the screen is the result of electrical fluxes between the mouse, the mainboard (and other circuits) and the screen. These fluxes can be represented as a flow of zeroes and ones under a machine language structure. But the reality is that normally nobody is interested in making that representation explicit. Anyhow, no matter what the intention of the user is, the actual flow of electrical flux runs through the circuit and that is what counts as, for example, the pointer of the mouse.

This series of code in its two aspects is properly entangled with the former two series. It can be said that it is in accordance with codification that the series of energy flux is pre-formed and that the series of circuitry is configured. Thus, the series of code is structured, as schemed in flowcharts. It is by means of the series of code, direct machine language, that we can speak of software, undoubtedly. Still, this code series would be nothing without the energy series and the circuitry series. The pre-formation of energy consumption, the configuration of circuitry and the structuration of software all converge logically.

We accept that the first two series respond to a formal distinction whose conceptual gain does not appear explicit. Nevertheless, we consider that the formal distinction of the first two series is as important as the understanding of the specificity of the code series for the whole ontology of software. The idea of claiming that software is only information, meta-data or matter, does not seem to be enough. Enough for what? For a consistent frame to interpret digital technology in modern society. But before addressing this question, at least briefly in this article, we propose to examine the second dimension of software, the structural dimension.

These three series could be misunderstood if they were taken as two material series versus a symbolic series. All three series have their own level of information. Although previous examples leave the impression that the two first series have a material nature while the third is just symbolic, it is important to note that a proper functioning of the electrical series requires different grades of information in accordance with the different voltage levels. Equally, the series of the circuitry implies some symbolization that assures that code will run properly. Finally, a series of code contains material elements as well as energetic elements. To explain this in other words, each of the series necessarily possesses a serial duality mainly because each of them requires an informational series of its own to which energy, circuits and code are linked.

This is the first dimension of the series that lets us interpret software as a process in which circuits, energy and code converge. This first dimension requires, nonetheless, to be completed by the user. Inasmuch as any software runs itself, it is essential to include the dimension of the human action. This dimension would include at least three components: the body, intentionality and the computational abilities of the user. This article only studies the serial and the structural dimensions of software, not its human dimension[11]. After developing these serial and structural dimensions we will be able to address the human action in and with software.

3.2. Structural Dimension of Software

Besides the previous ontological description, we can now address the structural dimension of software. This dimension primarily refers to the abstract structure any software would respond to. Besides the various levels of code and its functioning, the serial dimension has shown, the series of code can be understood in a logical simple way: a relation between commands, variables, calculations, inputs and outputs. This structural dimension will let us comprehend the reach of digital technology and its possibilities, and the correct way to introduce the idea of a digital universe.

A command in any computer language is, strictly speaking, a series of more basic commands that the computer executes as a way of functioning (using circuits to let an energy flow by). In the long run, any program in a particular computer language could be itself into a command on a higher-level language. This has happened many times (pointer, windows, sprites, and the like). Any subroutine or objects of programming, to which programming is oriented, are commands or sets of commands that the high-level programmer ignores and does not need to know or be aware of.[12] We can classify the commands in three types: input or information capturing commands, processing or calculation commands and output or presentation commands.

Variables should be considered in their relation to memory addresses. There we keep or store any kind of alphanumerical registry (permanently or temporarily). Just as a command, a variable is an essential condition for the full operability of any computer. A registry can be allocated in a memory address and then retrieved for any computation. We can distinguish here two levels: the physical address and the variable in code. Variables are

[11] Human intentionality will guide this analysis. On the one hand, Juan Manuel Jaramillo Uribe (2020) recently argued that the application of intentional explanation to theories that give account of human products, more than fruitful; on the other hand, Juan Carlos Moreno and his team has recently synthesized how the problem of human agency in technics is key to properly give account of technological reality (Moreno Ortiz *et al.*, 2020). We expect to articulate this fourth series in a general ontological theory of software.

[12] Probably the illusion of software as independent from energy and circuits emerges from the storage media (a part of the circuitry series), for it creates the illusion as if we were carrying code. But we must not be confused because on this storage medium we have a kind of registry of something that counts as code that becomes software once it is executed on a computer by a user. In this, sense, we shall keep in mind that software is pure actuality

traditionally distinguished between numeric and alphanumeric, i.e., those that can serve to do mathematical computations and those that simply store chains of characters.

In turn, chains of variables can be established, as in databases. But variables do not only have this usage. The content of variables can be combined and/or transformed in other contents to be kept new or the same variables. This last aspect takes us to the computation or the mathematical processing of variables through commands.

The commands carry out computations with the assigned content of variables. Computations are, in principle, simple processes and they work in blocks. The trick of any computer code is to understand that it makes a complex set of computations in order to obtain a specific result from specific data, depending on the sequence of commands and the kind of variables defined. We can speak of a hierarchy of computations in accordance with the whole of the computer code programmed.

Let us now describe the relationship between commands, variables and calculations. Variables are determined to be filled with data.

The main issue at stake in this regard is that it is necessary to fill the variables with data, which means storing values in a physical memory address. This is what is done when the user, for example, types in numbers (or data) to compute. In order to capture the information given by the user, it is necessary to implement a capturing command, which can catch any input. In fact, a user other than the programmer does not know what specific information is being expected for her to introduce unless the program itself asks for specific information. In this sense, we can identify several commands for making presentations on screen. Information introduced in this manner is an aspect different from data processing, although the computer will process them like any other data.

The results of data processing can be presented on the screen, if necessary. The lapse of time between the input or capture of data, the processing and the presentation of the result determines the complexity of the program. In this sense, we can understand that software has a limited time for proper computation: an inferior or internal time in function of the data input and the intermediate time in function of the time processing requires.

This basic structure in which variables, order and syntax of the program, processing of variables, the capture of data in variables and presentation of data seems to give a complete structural account of any code. This chain of commands is what constitutes the identity of the program, i.e. do what is supposed to do.

IV. Software and Digital Universe

We can reach a conclusion from the previous reflections regarding an ontology of software, at least in these first two dimensions. We will consider mostly the importance of the perspective of the digital universe as the plane of immanence of all phenomena related to the computational reality in nowadays society.

Any computer program or software always works singularly, i.e. in a singular machine. Once Robert Elliot Kahn and Vinton Gray Cerf achieved the development of the TCP/IP protocol in 1989, and the University of California lent it to Public Domain, singular machines could be steadily connected and, with time, they would be spread all around the globe. Let us not forget Ted Nelson's conception of the Hypertext and Hypermedia that actualized the first Internet media. Like these, many other software developments are what made the Internet possible. Under this perspective, the Internet emerges not as a particular software, but as a digital set (in contrast with Simondon's technical set (Simondon, 1989)), or a set of programs running altogether.

As we have pointed out, Floridi's approach to the philosophy of computing and information places information as the ground for the digital phenomena. But, under an ontology of software, information is at best understood as a component of software. We cannot doubt that information is found in each of the series, but, as we have seen, we found different levels of information in each one of them. The structural dimension of software implies a logical configuration as we have mentioned above. But we shall not miss that the structure described is external to the information. Anyhow, the real problem starts if we try to understand the specificity of information or if we try to make explicit what we may mean by information.

We possibly need to develop a critique of information in order to understand what we mean by it[13]. From this ontological perspective, even without such a critique, we can consider that information is digitized (in the convergence of the three series), which means that a software operates as a translator, and thus that information is placed on a new plane of immanence. The important feature in this view is that digitalization enlarges the amount of digital data, making possible information as such[14]. The process of

[13] Discussing Floridi's entire conception of information, especially the idea of infosphere in contrast with the idea of a digital universe, overflows the possibilities of this article. Nevertheless, such contrast needs at least the preliminary remarks we pretend with this article.

[14] The idea of "virtual reality" is explicitly avoided as we explained elsewhere (Maldonado Serrano & Rodríguez, 2018)

digitization itself constitutes something else, a transcendental space we propose to call the digital universe (Maldonado Serrano & Rodríguez, 2014).

But, the connections through the internet open to us a universal space that we can understand only as a transcendental space or milieu. Nevertheless, this universal connectedness is not a possibility, it is rather, a reality. So, the transcendental task of philosophy would be, just as Deleuze (1990) argued, to study the conditions of reality of this transcendental milieu, different from the transcendental philosophy as the study of conditions of possibilities of experience. Philosophy of software is transcendental empiricism in this sense.

References

Berry, D. M. (2011). *The Philosophy of Software: Code and Mediation in the Digital Age*. London, England: Palgrave Macmillan.

Colburn, T. R. (1999). Software, Abstraction and Ontology. *The Monist*, *82*(1), 3-19.

Deleuze, G. (1990). *Pourparlers*. Paris, France. Les Editions de Minuit.

Feenberg, A. (2002). *Transforming Technology: A Critical Theory Revisited* (2nd ed.). Oxford, England: Oxford University Press.

Floridi, L. (2010). *Information a Very Short Introduction*. Oxford, England: Oxford University Press.

Floridi, L. (2011). *The philosophy of information*. Oxford, England: Oxford University Press.

Heidegger, M. (2010). *Being and Time* (J. Stambaugh y D. Schmidt, Trads.). Nueva York, USA: State University of New York Press.

Hui, Y. (2016). *On the Existence of Digital Objects* (1.ª ed., Vol. 48). Minnesota, USA: University of Minnesota Press.

Koepsell, D. R. (2003). *The Ontology of Cyberspace: Philosophy, Law, and the Future of Intellectual Property*. Colorado, USA: Open Court Publishing.

Maldonado Serrano, J. F., y Rodríguez, D. A. (2014). Humanidad y universo digital: Prolegómenos al problema ético de la utilidad y el perjuicio de lo digital para la vida. *Análisis*, *46*(48), 27-40. doi: 10.15332/s0120-8454.2014.0084.02_

Maldonado Serrano, J. F., & Rodríguez, D. A. (2018). Critical digitality: From the virtual to the digital. *Praxis Filosófica*, (45S), 145-163. doi: 10.25100/pfilosofica.v0i45S.6134

Monterroza Ríos, Á. D. (2018). *La naturaleza heterogénea de los artefáctos técnicos: Un análisis ontológico*. Medellín, Colombia: Fondo Editorial ITM.

Moreno Ortiz, J. C., Fonseca Martínez, M. A., Prada Rodríguez, M. L., Orrego Echeverría, I. A., Pérez Jiménez, J. A., & Rengifo Ariza, L. E. (2020). *Tecnología, agencia y transhumanismo*. Bogotá, Colombia: Universidad Santo Tomás.

Simondon, G. (1989). *Du Mode d'Existence des Objects Techniques: Edition augmentèe*. Paris, France: Aubier.

Stiegler, B. (1994). *La technique et le temps: La faute d'Epiméthée*. Paris, France: Galilée/Cité des sciences et de l'industrie.

Stiegler, B. (1996). *La technique et le temps: La désorientation*. París, Francia: Galilée/Cité des sciences et de l'industrie.

Stiegler, B. (2001). *La technique et le temps: Le temps du cinéma et la question du mal-étre*. París, Francia: Galilée/Cité des sciences et de l'industrie.

Jaramillo-Uribe, J.M. (2020). El enfoque intencional en las ciencias sociales: Una mirada estructuralista de las teorías científicas intencionales. *Praxis Filosófica*, (50), 141-160. doi: 10.25100/pfilosofica.v0i50.8783