Topoi (2016) 35:203–212
DOI 10.1007/s11245-014-9248-5

What Can We Know of Computational Information? Measuring, Quantity, and Quality at Work in Programmable Artifacts

Federico Gobbo · Marco Benini

Published online: 9 April 2014

Abstract This paper explores the problem of knowledge in computational informational organisms, i.e. organisms that include a computing machinery at the artifact side. Although information can be understood in many ways, from the second half of the past century information is getting more and more digitised, von Neumann machines becoming dominant. Computational information is a challenge for the act of measuring, as neither purely quantitative nor totally qualitative approaches satisfy the need to explain the interplay among the agents producing and managing computational information. In this paper, Floridi's method of levels of abstraction is applied to the analysis of computational information, with a chief interest in the concepts of information measure, quantification and quality.

Keywords Informational organism · Computational information · Computational complexity · Information quality · Information measuring · Quantitative measuring

Parts of this paper were presented at the Fifth Philosophy of Information Workshop at the University of Hertfordshire, when Marco Benini was a member of the School of Mathematics, University of Leeds, UK, while Federico Gobbo was appointed by the University of L′Aquila. The academic affiliations of both the authors have changed since then.

F. Gobbo
University of Amsterdam, Spuistraat 210, 1012VT Amsterdam, The Netherlands
e-mail: F.Gobbo@uva.nl

M. Benini (✉)
University of Insubria, via Mazzini 5, 21100 Varese, Italy
e-mail: marco.benini@uninsubria.it

1 Introduction

Measuring is a distinctive feature of Engineering and it is often characterised as 'describing a phenomenon in terms of numbers'. This is also the common usage of the word: sometimes, it is used even as a synonym for 'quantification'[1]. Nevertheless, in computer science (CS), this meaning is not always respected. For example, the measuring of computational complexity, an essential part of CS, cannot be described by usual quantification, but on the contrary it needs a more subtle concept. On the other hand, Floridi (2013) has pointed out that quality, as intended in its usual meaning—i.e., a distinctive attribute that distinguishes the object from the standard level—fails to be a satisfying concept in our contemporary world, where information and communication technologies (ICTs) became pervasive. Beavers notes that "the digital revolution, which began with the popularization of the personal computer around 1980, inaugurated an era in which people appear in consort on an extended network intermixed with other, non-human information processors, all 'inforgs' to use Floridi's term" (Beavers 2011, p. 264). For this reason, the Onlife Manifesto The Onlife Group (2013) states that the changes driven by ICTs force us to re-engineer our concepts, especially the ones related to information, among the others, measuring, quantity, and quality. Reality changed so quickly with ICTs that we risk to be conceptually wrong-footed in making sense of reality and in our interaction with it: the Onlife Manifesto's programme addresses the re-engineering of the conceptual tools we use to understand the relation among human beings, ICTs, and reality. In this paper, we will address the re-engineering of

[1] The term is used in this paper in the generic sense, not in its logical meaning, as it does not refer to quantifiers.

measuring quantity, and quality in the case of computational information. In other words, we discuss how these actions should be redefined so as to give meaning to a world more and more shaped by ICTs.

From now on, we will adopt the view of Informational Structural Realism, defended by Floridi in many publications, e.g., (Floridi 2011, ch. 15), as our conceptual framework. Although "'information' can be understood in many ways, e.g. as signals, natural patterns or nomic regularities, as instructions, as content, as news, as synonymous with data, as power, or as economic resource, and so forth" (Floridi 2011, p. 226), von Neumann machines are typically used to produce, store, and manage information. The von Neumann machine (VNM) is by far the most influential architecture of digital computers in history, widely applied everywhere, from supercomputers to layperson's smartphones. From now on, we will consider only artifacts based on this architecture, leaving other forms of computing out of our analysis: not because analogue computing, quantum computing, and other models are not of interest, but simply because VNM covers the large majority of real cases. The reader should be warned that, in principle, our analysis can be extended to other computing architectures but, as remarked in Sect. 4, these extensions are not necessarily immediate or straightforward.

Within VNM-based devices, the ways information is understood eventually share the same nature: signals, natural patterns, news, and so forth are expressed in terms of numbers, and therefore they can be considered computational information. It is important to notice by now that numbers perse carry no information. That is, computational denotation does not comprehend its semantic content.

Therefore, to re-engineer the concepts of measuring, quantity, and quality, the process of sense-making of computational information should be taken into account. This process, in its most elementary form, has to build a description of the complex structure of knowledge which enables to interpret the behaviour of an informational organism (inforg). The scope of the analysis pursued in this article is limited to the aspects directly related to programming, as this subject poses some concrete philosophical problems that mark the essential differences from the engineering tradition when measuring quality and quantity inside a computational inforg based on a VNM (by now, *c*-inforg). The description resulting from the sense-making process may be either internal or external: in the former case, it provides an account of how the various components of a *c*-inforg, notably the computational artifact and its human counterpart, interact with each other; in the latter case, the description focuses on how the *c*-inforg, seen as a unit, is perceived and understood in the environment it operates in.

In the context of programming, many initial analyses of the external view can be found in literature: significant for the proposing approach is Gobbo and Benini (forthcoming), which discusses to what extent programming can be understood as a process detached from the human counterpart that wrote it, considering how a program is used and maintained during its life-cycle, becoming part of different *c*-inforgs. Similarly, Costantini and Gobbo (2013), building on the paradigm of agent-oriented programming, see Shoham (1990), study the purpose of programmable artifacts in multi-agent systems.

In the present work, we will address a few critical problems in the description of programs inside a *c*-inforg—thus adopting an internal view—namely the measuring of quantity and quality, when the 'quantity' to measure is computational complexity, or when the 'quantity' is naturally expressed by means of a real number. It is evident that these cases are significantly present in most programs, as efficiency and numerical computations are fundamental aspects in programming. To allow the discussion of these cases, a suitable framework has to be developed, as a set of notions and terminology, which is done in Sects. 2 and 3. It turns out that, appropriately using this apparatus, the proposed cases show a counter-intuitive behaviour that forces us to reconsider—to re-engineer—the way we are traditionally used to understand and present these examples, and the problems in which they are embedded. To confirm that the achieved results help to attain depth in the understanding of CS, an experiment in teaching has been successfully conducted: it is reported in Sect. 5.

So, the paper is structured as follows. In the next Sect. 2 programmability, being the distinctive property of VNMs, will be addressed, to explain the special features of computational information in the analysis of inforgs. In Sect. 3 the method of the levels of abstraction Floridi (2008) is applied to *c*-inforgs, using an informational structuralist approach, with some examples. Then, in Sect. 4 the measuring of information in the case of *c*-inforgs is analysed via the two critical examples mentioned above. Section 5 presents a preliminary case study of the method applied in practice, in teaching teachers of informatics and CS within a lifelong learning programme. Finally, Sect. 6 presents the conclusions and further directions of work.

2 The Consequences of Programmability

The distinctive property of VNMs is their programmability. In his seminal paper on the architecture that carries his name, von Neumann (1993) states that the "instructions which govern this operation [of calculation]" can be physically expressed in many forms (some original examples being punchcards, teletype tape or photographically impression) but always requiring "some code to express the logical and algebraic definition of the problem under consideration, as well as the necessary numerical material". The definition and use of this code is the property we now call programmability. As underlined by the philosophers of CS such as Turner, programs appear to have a dual nature Turner (2013): the source code (*textual* nature, human-readable) and the object code (*binary* nature, machine-readable). This property has an important consequence: either the source code of the *c*-inforg is *open*, i.e., it can be inspected and eventually modified by the human counterpart of the *c*-inforg, possibly a programmer, or conversely its textual nature is inaccessible, and therefore the human counterpart can be a (power) end-user but not a programmer by definition, as nobody can access and modify the source code. VNMs are the technical artifacts of *c*-inforgs. In Turner's terms Turner (2013), the structural properties (how artifacts are made inside, their physical and symbolic makeup) of closed *c*-inforgs are unavailable, while the functional properties (black box specifications, in terms of inputs and outputs) can still be inspected. In the terminology by Gobbo and Benini for the analysis of history of computing, some information is hidden: in this case, the textual nature of code is closed to inspection Gobbo and Benini (2013). What are the consequences of programmability, in particular as it is related to knowledge?

Floridi recently proposed the concept of *in-betweeness* to define the basic functional property of a technology used by an agent Floridi (2013). In the simplest case, first-order technology is in-between human agents and nature. For instance, a pair of sunglasses is in-between the wearer's eyes (the human side of the inforg, or simply the user) and sunlight (natural phenomenon, in general the *affordance*). Clearly, VNMs are not first-order technologies.

Second-order technologies put themselves between human agents and other technologies, while third-order technologies put themselves between technologies and technologies. One example is a key that is a technology between the user and the lock. Moreover, the lock itself is a third-order technology, being in-between the key and the door. VNMs can be second-order or third-order technologies as well. For instance, when we browse a web site, we are using a second-order computer where we are typing the desired web address, while the obtained information is caught from a server, where the web site is hosted; no human being is involved in this process, but only computers 'talking' to the others. For example, the simple act of browsing is a combination of second-order and third-order technologies: the browser is in-between the user and the act of browsing (second order technology) while the operating system, the tcp/ip protocol, drivers, hardware, etc. are in-between themselves, not requiring an interaction with the user (third order technologies). The existence of computational third-order technologies is due to the desire of information hiding: an end-user does not want to know the client–server architecture to perform a simple action as browsing the web, since the interest is in the content presented by the web site, not its representation in the local computer or the net.

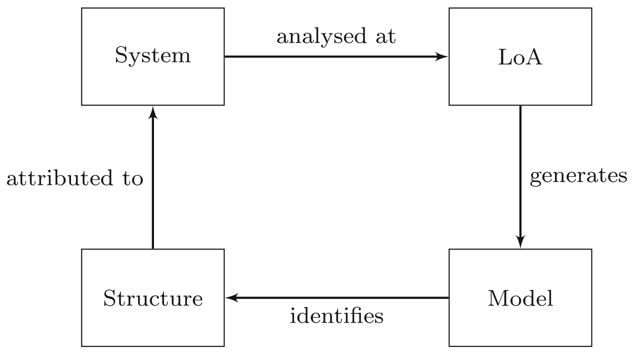
$$\begin{tikzcd}
\boxed{\text{System}} \arrow[r, "\text{analysed at}"] & \boxed{\text{LoA}} \arrow[d, "\text{generates}"] \\
\boxed{\text{Structure}} \arrow[u, "\text{attributed to}"] & \boxed{\text{Model}} \arrow[l, "\text{identifies}"]
\end{tikzcd}$$

Fig. 1 The SLMS scheme, adapted from (Floridi 2011, p. 349)

A *c*-inforg is ultimately formed by (at least a) human agent and a (non-empty set of) engineered VNMs. Together, they form a chain of third-order technologies with a second-order technology at one end.

3 System, Level, Model, Structure

The method of the levels of abstraction (LoAs) is the conceptual toolbox apt to analyse inforgs Floridi (2008). Within this method, a system cannot be inspected directly, but only through the analytic lens of LoAs. Moreover, a LoA presumes the existence of *observables*, i.e., typed variables that model a given feature. Observables constitute the objects of the model, which is generated by the level itself, that identifies the structural properties (in the sense above, i.e., how the system is actually made up) of the system. The theory makes sense of the system through the finite set of LoAs, their generated models that identify the structural properties (Floridi 2011, pp. 347–352). Figure 1 depicts the system–level–model–structure schema (SLMS).

When observables are discrete, they may be ordered. In this case, a gradient of abstraction (GoA) can represent a stack of LoAs so as to facilitate parallel analysis: more lenses worn at the same time. However, it is important to note that observables are not necessarily quantitative nor empirical. For instance, the statement that the Greek goddess Athena was born from Zeus is a qualitative observable that is completely acceptable if put in the right LoA, about Greek mythology and culture, the system of reference (Floridi 2011, p. 49). The properties of observables determine the characters of the LoA they are in. The properties of observables are central in re-engineering the concepts of quantity, quality and measurement in the case of *c*-inforgs, as we will see in the Sect. 4. Before doing that, it is necessary to present briefly the method used here. Each LoA is

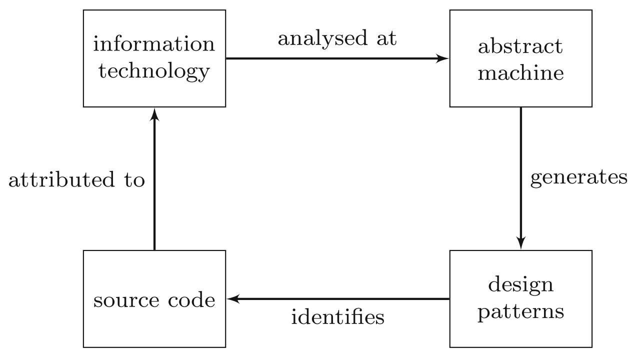

Fig. 2 The SLMS scheme applied to a programmer at work

paired with a corresponding level of organisation (LoO) of the technical artifact (Floridi 2011, p. 69).

Let us take the operating system in-between the user and an external drive as a relevant example (we will omit some fine-grained LoAs for simplicity, taking the point of view of a non-expert end-user). In a modern VNM-based computer, the operating system is a software in-between the user and the hardware of the computer. The LoA of the operating system[2] is formed by the observables that the user can manipulate (e.g., the actions needed to unplug a USB flash drive) while the corresponding LoO contains the hardware pieces devoted to control the plugging of USB flash drives, and the drives themselves. Note that some details of the technical artifacts are not pertinent at the level of some particular LoA, and therefore they will be negligible also at the corresponding LoO. Following the example, the actual shape of the USB flash drive (key, pen, or anything else) does not have a role in the plugging, at the LoA of the operating system, even if it may have a role at a different LoA, e.g., at the physical layout, since the size of the device (an observable) must match the size of the plug. On the other side of the LoA/LoO pair, the LoO is no more a mere description of the hardware, rather it is the ontological counterpart that permits the epistemology behind the LoA.

However, the LoA/LoO pairs are impossible to be defined if there is no purpose involved. Different purposes can be identified for the operating system, depending on the human agents involved: programmers, end-users, system administrators, market analysts, software vendors can have very different purposes, even in contrast one to the other. Purposes, along with their scopes and their how-to instructions, form the levels of explanations (LoE) in the method of the levels of abstraction (Floridi 2011, pp. 69–70). LoEs are epistemological as LoAs, also being in the human agent's mind, but they are explictly telic. In sum, to apply the SLMS schema we need to identify the pairs of LoAs and LoOs and the LoEs applied over them by the human agents.

Let us give some examples. As explained before, programming is the distinctive feature of a *c*-inforg. What is the LoE of a programmer during the act of programming? The simplest case is when the programmer does not work in team and develops the program from scratch—this is rarely the case nowadays in real life, but let us keep the example simple for clarity. For a prototypical programmer, the system is an information technology to be analysed by a tower of abstract machines, where 'tower' means an ordered stack of abstract machines, and each machine corresponds to a LoA. The choice of a programming language determines the kind of observables admissible in the LoA: for example, a procedural programming language implies a different LoA compared to an object-oriented one. Once the LoA is defined, a model of the program is thought by the programmer, following the design patterns available. Design patterns are best practices in programming which the programmer takes into account in writing the actual code, especially in the case of large projects—see at least the classic book by Brooks (1995). Considerations about algorithms and executions are important, and also about how the program should be organised internally to facilitate maintenance. This model is used to identify the actual source code that is developed by the programmer that constitutes the structure of the whole theory to be attributed to the system. Figure 2 shows the LoE of the working programmer.

On the other hand, end-users have different perspectives compared to programmers. We can explain how an end-user relates to a computer through a desktop application. An end-user is a human agent who is not able to program nor interested in learning to do it, or a programmer who pretends not to be able to program, for certain purposes—e.g., testing. The LoE is simple: I follow some procedures, and the machine will give me the expected output. In Turner's terms, end-users are interested only in the functional properties (black box), not in the structural ones: they know the existence of the latter, but they want that such kind of information remain hidden. For simplicity, let us isolate the *c*-inforg: the end-user and the computer are not connected to the Internet, as in the old days, when desktop applications were made for the self-production of documents such as printed letters or spreadsheets. These productions are instances of the end-user's LoE.

The desktop application is the second-order technology in-between the end-user and the black box, the 'affordance', in Floridi's terms Floridi (2013). The desktop

[2] Here and elsewhere, we use the common expression "the LoA of some object": this is a short way to express the right concept of "the LoA associated to some object as in the mind of the describer/writer/observer/agent" where the context identifies the human being involved.

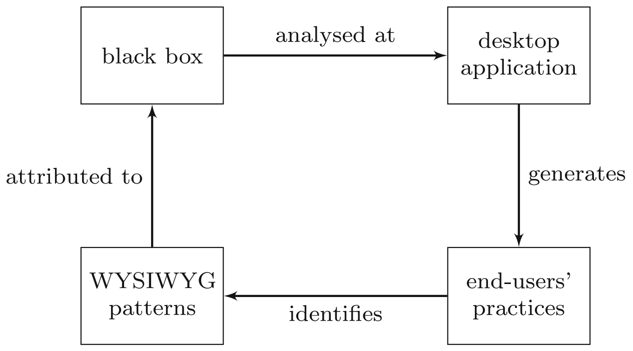

Fig. 3 The SLMS scheme applied to an end-user's goal

application acts as the LoA to perform the goals expressed in the LoE: for instance, if the home budget is the LoE, the spreadsheet will be a reasonable LoA for the end-user. Suppose that the end-user is not skilled with the spreadsheet: either following a trial and error strategy ('guess, click and see what happens') or reading the manual, the practices generated by the use of the desktop application will constitute the model of the 'theory': the end-user will identify some best practices—as well as what not to do!—to achieve the desired goal. It is natural for the end-user to identify some common patterns in the graphic user interface. For example, if the current value of investment is needed, the best practice will be something like: 'after selecting the relevant records, from the menu Insert, click Function; a pop-up window will appear; select Finance from the menu Category and scroll down until you find the desired function; select it, then click the OK button'. Of course, soon the end-user will learn that all the financial functions will be found therein, so that a pattern 'how to find a financial function' can be extracted. The above identified WYSIWYG patterns in the graphic user interface form the structure of the 'theory' of the end-user. Figure 3 depicts this example.

It is clear that an open *c*-inforg, where the source code is available to inspection, is potentially far more informative compared to a closed *c*-inforg. However, thanks to the preliminary data of the case study presented below (Sect. 5), we are confident that the application of the method of the levels of abstraction to closed *c*-inforgs is appropriate too, as it helps to reason about complex realities with a rigorous method, where validity is clearly and locally defined by the scope of the appropriate LoA.

The method can be applied to complex cases too. For example, we can identify the user profile in Facebook as a closed *c*-inforg. At this LoA, Facebook is second-order technology, put in-betweeen the real user (the human agent) and the social network, i.e., the 'affordance', filtered through Facebook 'friends', i.e., other Facebook user profiles. Moreover, the LoO can reveal trends or even biases in the real user's preferences or inclinations (part of the LoE). For example, Alice prefers to access her Facebook account with her smartphone when she is walking around, during free time, while Bob is used to access his user profile through his laptop to have a pause from work, because he works as a ICT consultant. The different LoOs (smartphone vs. laptop) have important consequences in the actions normally performed by the inforgs: probably, Alice will be more inclined to share photos and microblogging posts, while on the contrary Bob will be keen to share interesting web sites or write longer blog posts on his Facebook wall. The fact that the method clearly distinguishes levels of abstraction (LoA), organisation (LoO) and explanation (LoE) allows to analyse complex phenomena without implicit ontological or epistemological commitments, so that knowledge can be well defined in a transparent way—a claim widely discussed by Floridi in his works. This is especially relevant in the case of closed *c*-inforgs, where most information is hidden.

4 Measuring Computational Information

For the layperson, computers deal only with digital numbers and thus there is only quantitative information in them. But, as we have seen, a computing machine is constructed and programmed by conceiving it as a complex network of LoAs and LoOs, where the computational numbers are just the observables in the lowest level accessible to a programmer. Along this sequence of LoAs, computational numbers assume meanings—when they do not get hidden—that are completely different and apparently unrelated to their numerical and quantitative nature. It is worth describing how the process of assigning a meaning to computational number takes place, what does it imply. In order to do so, we focus on two examples, preceding their presentation with a discussion on the nature of computational numbers.

4.1 Computational Numbers and Structure

A basic fact in CS is that each piece of data is represented by a computational number, at least at the most basic level of VNMs—although this fact can be extended to other computational models, this may be complex and delicate, and out of the scope of the present paper. Here, data and information are distinct, as already remarked in the Introduction: what provides a meaning to data is the LoA (and the involved LoEs) at which they are considered, as explained before, thus the LoAs allow to interpret data as information. LoAs are inherently linked to encodings: to say that a piece of information in a LoA is represented by an observable in the corresponding LoO, means that the LoA/LoO pair maps down to the VNM via some encoding, eventually as a result of the composition of other encodings associated to LoA/LoO pairs at a lower level of abstraction.

From a purely mathematical point of view, encodings are possible because computational numbers possess a rich and complex structure[3]. In fact, they are numbers, subject to the operations and properties of arithmetic—and arithmetical operations are part of what the VNM [specifically, the central processing unit of a computer, as introduced by von Neumann Burks et al. (1946)] provides to the programmer; but computational numbers are also vectors of Boolean values, subject to the operation of Boolean algebras, like conjunction, disjunction, etc.—and, again, logical operations are part of the instruction set of a VNM, see, e.g., Hennessy and Patterson (2006); moreover, computational numbers are strings of bits, subject to operations on strings, like concatenation, rotations, shifts, etc.—again, these operations are explicitly realised by the VNM's hardware, both as instructions in the machine language, or in the organisation of memory as a sequence of numbers of fixed size, see, e.g., Hennessy and Patterson (2006). It is worth remarking that, although other computational models exhibit similar characters, their operational semantics may be very different because their operations are performed in a diverse context.

Thus, computational numbers are explained by a number of rich mathematical structures, each one supported by a mathematical theory that provides many properties. So, although the LoO of a VNM exposes just computational numbers as observables, the corresponding LoA has many instructions to manipulate the numbers according to different interpretations, one for each of the mathematical structures cited above. In turn, each of the mathematical theories about these structures becomes a LoE for the VNM LoA, explaining its way to compute in the context of a theory and providing a purpose for each instruction.

By using the properties of computational numbers and mapping them to the properties of the desired data, a programmer creates new LoAs: for example, a character is encoded as a number, e.g., by using the ASCII table, and, thus, a string becomes a sequence of numbers in the memory. Here, the programmer uses equality of numbers to simulate equality of characters, and the sequential organisation of memory to simulate the fact that a string is a sequence of characters. Operations on strings are implemented mapping them to operations on the computational numbers representing the string in such a way that the properties of the string, i.e., their LoE, become, via the encoding, true properties of the representing numbers in the LoEs over the VNM.

Thus, as said before in Sect. 3, the programmer constructs a tower of abstract machines, or equivalently LoAs, on the top of the basic VNM. Encodings between these machines are needed to allow the VNM at the bottom to compute[4], as they associate entities of the abstract level to computational numbers. A matter of fact is that the higher levels in the tower are often explained by LoEs of a non-strictly mathematical nature, as is the case of trust or security in web applications. The possible non-formal nature of LoEs in higher LoAs poses the problem of how to justify the correctness of software, and, more in general, how to justify the properties—the purposes, as illustrated before—of programs with respect to the LoEs adopted by the user of the program. Similar issues arise also when considering properties, like efficiency, describing not what has to be computed, but how the computation takes place: in fact, in logical terms, these are higher-order properties as they do not refer to the objects of the discourse, but rather to a structure over objects, like computations in the case of efficiency.

In the following, we want to describe two real problems in CS where measuring, quality and quantity are words whose meaning has to be interpreted in a strictly different way from the common sense. And, as we will discuss, this claim is true because the involved LoAs require LoEs which are structured considering that we are operating inside a *c*-inforg, thus with the presence of the computational artifact, and, consequently, to allow the possibility of an encoding.

4.2 Measuring Computational Complexity

The theory of computational complexity [see Papadimitriou (1994) for a detailed survey] is the branch of Theoretical Computer Science which studies the efficiency of algorithms in terms of time, i.e., number of steps to perform a complete computation, and space, i.e., the quantity of memory cells which are needed to obtain a result. An algorithm A is considered more efficient than B when it

[3] Mathematically, this structure is the one of integers modulo 2^n, represented in the binary notation. The usual integers do not possess the structure of vectors of Booleans, and the reals are not even representable, as discussed in Sect. 4.3.

[4] An interesting remark posed by an anonymous reviewer is whether a LoA is needed for the VNM to compute. Here, is evident that a LoA is strictly needed: the computation really performed by a VNM has no meaning unless understood as the result of the encoding of the abstract LoA into the "real machine". For example, the operation $x + 32$ at the VNM level increments the number x by 32, while, in a LoA which codes characters using the ASCII table, the same operation converts upper case letters into lower-case (this example comes from the standard library of the C language, so it refers to real code)—and it is "correct" only when x is an upper-case letter, otherwise one has to accept, e.g., that the lower-case form of 1 is Q, and that the lower-case form of z does not exist.

uses less the resource we want to minimise, either time or space.

But there is problem which is not irrelevant: the time (and the space) needed to perform a computation depends on the input. Researchers have abstracted over the input by considering only its size: so choosing between two algorithms A and B means to consider the worst performance of an algorithm A for an input of size n, the worst performance of an algorithm B for an input of the same size, and to extend this comparison to every n. Thus, A is certainly more efficient than B if, *for every* n, the time A spends to compute on the worst input[5] of size n is less than the time B uses to perform the same task on its worst input of size n.

Evidently, this notion of 'more efficient' is not adequate: in most cases, algorithms are incomparable, as each one may be more efficient than the other for some values of n. But, researchers observed that what really matters is the long-term behaviour of an algorithm: when n is 'big' which algorithm under consideration performs better? The fundamental fact to observe is that we can consider the function which maps n to the worst timing performance of an algorithm with an input of size n. What we consider as the *complexity* of the algorithm is exactly this function. And, when we ask to compare two algorithms, we really compare their complexity functions; the long-term behaviour is then captured by saying that one function *definitely dominates* the other one, using the jargon of mathematical analysis. In this way, computer scientists measure efficiency of algorithms using functions instead of numbers.

But this produces a qualitative change in the classification of algorithms: the complexity function associated with some algorithms may exhibit a behaviour that is similar to the one of a polynomial; in other cases, the complexity function grows much faster, as the exponential function or even more. This observation introduces a 'qualitative' distinction between algorithms: the ones which are 'feasible' (polynomial) and the ones which are 'practically uncomputable' (exponential). This distinction is codified in the so-called *extended Church-Turing thesis* which says that the only practically computable problems are the ones admitting a polynomial algorithm able to compute them—a discussion of this thesis and its relevance in CS can be found in any textbook about computational complexity, e.g., Papadimitriou (1994).

The search for the borderline between feasible and unfeasible algorithms is still open: the class of polynomial algorithms operating on a non-deterministic Turing machine is the borderline, as all these algorithms can be easily simulated on a deterministic machine by trying all the possible non-deterministic paths, and this takes an exponential time, i.e., the complexity function of the simulated algorithm is exponential; but a subclass of those algorithms, the 'decision procedures', is particularly relevant. This class is called **NP** in literature and the problem if **NP** is distinct from **P**, the class of polynomial algorithms on a deterministic machine, is the most important and well-known open problem in Theoretical Computer Science, thus an eminently qualitative problem, see Fortnow (2013).

In sum, algorithmic complexity is a measure whose structure allows comparisons, via class membership and the taxonomy of complexity classes. Nevertheless, it is a measure which is intrinsically non-numerical, even if it expresses a sort of 'quantity'. Moreover, the essential problems in the structure are of a qualitative nature.

Rephrasing the previous conclusion in the framework of LoAs, we can say that the theory of computational complexity is a gradient of LoEs, analogous to the gradients of abstractions as introduced in Sect. 3: over the LoA in which the algorithm is implemented, there is a first LoE that provides a way to measure the number of steps the algorithm performs; over this LoE there is another LoE that summarises the measures at the first level into functions understood as algorithmic complexities; finally, the algorithmic complexities are classified according to a taxonomy whose purpose is to measure feasibility, e.g., with respect to the extended Church–Turing thesis, thus providing a third LoE. The way measures are linked in these three LoEs form an epistemological gradient from a quantity (number of steps) to a quality (membership in a complexity class), and a full understanding of the whole concept of computational complexity cannot but consider the whole gradient of explanations.

4.3 Equality of Reals

A large source of procedural solutions to practical problems in Engineering is Mathematical Analysis. Differential equations are the standard way to model most physical problems—a way that has proved to be extremely successful by the advance of technology in the last three centuries. With the advent of computing machineries, specific numerical methods have been developed to solve these equations, using the ability to quickly calculate over an enormous amount of data.

Since those methods rely on the ability to perform calculations over real numbers, we would like to represent these numbers inside machines together with the operations we are to perform on them. Unfortunately, such a representation does not exist. In fact, the cardinality of the set of real numbers is bigger than the one of natural numbers, as Cantor discovered in the second half of the nineteenth century. Since the cardinality of finite strings of symbols

5 We are illustrating the most widely studied notion, the one of worst-case analysis: there are also studies about average performances, but they are outside the scope of this paper.

over a denumerable alphabet is the same as the one of natural numbers, we do not have enough expressions to represent the totality of real numbers in a way suitable to computers' memories.

Of course, we can approximate real numbers by using rational numbers, which are representable. And this is the way programmers usually follows to simulate the analytical methods. But this choice introduces some problems. One of them, probably the simplest to understand, is equality: in fact, since every real number r gets represented by some number q_r, but the cardinality of rationals is strictly lower than the one of reals, it follows that there are at least two distinct real numbers r and s such that $q_r = q_s$. In mathematical terms, the structure of rationals is different from the structure of reals, and the mapping between the two of them is not preserving basic relations, like equality. So, in philosophical terms, two measures r and s may be the same for the machine, i.e., when $q_r = q_s$, even if they are different for a human being: this fact is a problem because the human being and the machine are both part of the same c-inforg, that is, the human uses the measures by means of the computational machinery.

To prevent these phenomena, mathematicians and computer scientists tried a different class of representation for real numbers. Instead of using a rational to approximate the value of a real up to some precision, the class of real numbers is restricted to the one whose decimal expansion can be generated by an algorithm. This class has the cardinality of naturals, and so it is representable; moreover, in most practical cases, the real numbers of interest lie in this class. Therefore, a representable real number becomes an algorithm generating the sequence of digits of the usual decimal representation. Suppose a and b are two representable real numbers, and suppose they are in the interval [0, 1) so that their integer part is 0. How can we compare them? The only way we have is to check their decimal expansions, one digit per step: at the n step, if $a(n) > b(n)$ then we can stop and say that $a > b$; if $a(n) < b(n)$, we can stop and say $a < b$; otherwise, if $a(n) = b(n)$ we must continue to the next step. Evidently, since real numbers are totally ordered, this algorithm will stop whenever $a \neq b$, and it will tell us which number is greater than the other. On the contrary, the algorithm is simply unable to say that $a = b$, as it requires an infinite number of steps to reach this conclusion. So, again, representable real numbers cannot be *exactly* represented, as their structure, namely equality, is not decidable, that is, outside the capability of computing machines.

In general, one can mathematically prove that no computable presentation of real numbers which preserves equality in the sense above is possible. The first philosophical consequence of this result is that measures which are valid in the human world are no more valid in the world of c-inforgs, since they lack a fundamental piece of structure, equality.

The second philosophical consequence is that what we called structure is not the usual, well-known reality of classical mathematics. Rather, it is the world of computational mathematics, which has its roots in the constructive approach to logic due to Brouwer Bridges and Vîţă (2011). While in the classical world, 'not different' is the same as equal, in this logical system, which is adequate for computing machineries, 'not different' and 'equal' are distinct concepts as the former refers to an infinitary computation which admits approximations, while the latter has no meaning *inside* the system, since it cannot be represented as an algorithm.

Mapping these two consequences in the framework of LoAs allows to understand them in a deeper way: in the case of computational real numbers, the LoE of real analysis is what justifies the algorithms, e.g., by showing how to numerically compute the solution of a differential equation, but the LoA limits the amount of possible operations, as equality is undecidable. In the terms of Sect. 4.1, this means that there is no possible encoding for equality to VNM's operations, i.e., the basic computational LoA. So, the LoE of real analysis is only partially adequate as it does not consider computability of equality an issue. Thus, the second consequence mentioned above says that there is a strict need for another LoE, the one of computational mathematics, to justify the encoding of the LoA of real numbers to the basic LoA of the VNM. Moreover, such a need for an additional LoE derives directly from the nature of a c-inforg, specifically from the presence of the computational artifact, which, by its nature, is unable to compute equality of reals, thus, strongly justifying Floridi's claim that the introduction of computing machineries has deeply changed the epistemological and ontological status of the world, as cited in the Introduction.

5 The Method in Practice: A Case Study

One of the authors, Dr. Gobbo, had the opportunity to test the method presented above in the classroom, in a special setting, in the period April–June 2013. All students were high-school teachers of informatics and CS, involved in a lifelong learning programme with qualified professors and teaching fellows at the local University of L′Aquila, to improve their skills and understanding of the discipline. The class was composed of ten members (four women, six men) previously selected by the Ministry all over the region of Abruzzo, Italy. Their qualifications and skills were already high, and in some cases their experience as teachers were long (more than 5 years), even if the goal of the whole programme was to prepare teachers with not so long experience in the high-school classrooms. The course was entitled *Informatica di Base* (in Italian, more or less 'Basics of Computer Science'), clearly an unattractive title for the students, who had some degree of freedom in choosing the courses; therefore, agreeing with the committee that lead the whole curriculum, the author spent eight hours (over 12 in total) presenting the method of the LoAs in class, proposing as the assignment to write a short essay (approx. 12,000 key strokes) on this part of the course. In the assignments, the model should be applied to explain a single topic of informatics (if practical) or CS (if theoretical) within their high-school classes; they should describe the LoA/LoO pairs and the LoEs (at least two: the teacher's and the students') of the topic, recording where they and the students found more problems, and possibly how they solved them.

At last, eight students chose to write the assignment, and seven of them were positive—in the non-positive assignment the point was missing, as no concrete case in the class was presented, but only a summary of the lectures given during the course. The topics were very different. Some assignments dealt with the art of computer programming, in defining the LoAs from the binary code, the compiler and the source code (mostly in C, C++ or Pascal) to software applications.

Their common problem was to define the correct LoE, especially the end-users', as neither the teacher nor the students had real interactions with end-users, but they could only imagine how end-users would react to their small programs. An assignment applied the model to the concept of browsing the web, helping the students to identify some LoAs they were not aware of, such as the TCP/IP protocols, and how to check the packets through a command-line interface—"something completely alien for them", as reported in the assignment. Another assignment dealt with the design and development of a web site for an association that would give an award (and a small grant) to the best project among the classes of their high-school. Actually they won, because they took "into consideration the users' needs, unlike other projects", as stated in the motivations of the award.

One of the topics that raised more interest in class was about the future of programming: will the distinction between power end-users and programmers become less and less evident? Although the perspective in class did not consider any futuristic speculation, there are already some trends showing that in the next future artifacts could be programmable easier than today. This could happen especially when the artifacts themselves are goal-driven, starting from their hardware—for instance, based on Raspberry Pi Upton and Halfacree (2013). In Floridi's terms, the re-ontologisation of artifacts is opening the door to the next stage of the information age: the 'onlife experience' (Floridi 2013, 8). However, history of computing shows that scenarios where pupils happily program their videogames in their tablets, like Alan Kay's Dynabook young users Kay (1972), are very unlikely to become effective, at least on a large scale. In fact, today, most young end-users play with their tablets without even being aware of the fact that their devices are programmable. Thus, we still argue that the observations presented in this paper will remain valid: for each power end-user that turns to programming, there will be an increasing number of end-user that will be satisfied to use what their devices provides them, disregarding programming as a way to customise or enhance their computational tools.

We know that a single class of ten students with only eight assignments is not significant in statistical terms. Anyhow, we think that this preliminary case study of application of the method presented in this paper can be a pilot for further explorations in this direction. Also, the practical experience shows how the methods supporting the authors' philosophical explorations can be fruitfully used to enhance the understanding of CS even in very concrete applications.

6 Conclusion and Further Work

Measuring is an aspect of information in CS that is quantitative and qualitative at the same time. As we have discussed and exemplified in Sect. 4, a measure carries a structure with itself which explains how it can be used. To be precise, we should say that a measure carries at least a different amount of levels of explanations—or even a gradient of explanation if a correspondent gradient of abstraction (GoA) is observable—following what has been explained in Sect. 3. The ultimate goal is to give account both of how to correctly manipulate the datum, and how to interpret the datum in the corresponding level in the gradient of abstractions, thus eventually describing its purpose.

Floridi's method of LoAs provides a very precise and transparent description of the epistemology of measuring, which is mostly significant in the case of *c*-inforgs. As depicted in Sect. 2, the computability of measures has a crucial role in *c*-inforgs' analysis.

The present work provides a sufficiently large landscape to justify the interest for further investigations of the presented concepts and of their mutual relations. Of course, the reader has surely noticed, many aspects worth deepening, have been just sketched: they are part of authors' plans for future developments. Nevertheless, the working philosopher of information cannot avoid the continuous interplay between quantitative and qualitative points of views of information.

Finally, the authors asked themselves whether their findings may help the concrete, real understanding of

Computer Science: in this respect, the experience described in Sect. 5 shows that the method of levels of abstraction can be usefully adopted in practice and, to some extent, part of this article has been inspired by the most interesting remarks and questions collected during that experience.

In this paper, the epistemological question (what can we know) has been addressed, so that an informed ethical approach to third-order technologies can be investigated, programmability being the crucial property to be taken into account. Computational information hiding raises interesting ethical issues; in particular, Floridi points out that "with the appearance of third-order technologies all the in-betweeness becomes internal, no longer ours but technologies' business […] we are depending on our technologies and our technologies are independent of us" (Floridi 2013, p. 115). However investigation of the ethical issues is out of the scope of this paper, and therefore it is left as a further work.

Acknowledgments We would like to thank all participants of the Fifth Workshop of the Philosophy of Information for their useful feedbacks to our work, and the reviewers for their compelling and interesting remarks. Also, Dr. Benini has been supported in this research by the project "Correctness by Construction" (CORCON), EU 7th framework programme, grant n. PIRSES-GA-2013-612638, and by the project "Abstract Mathematics for Actual Computation: Hilbert′s Program in the 21st Century" from the John Templeton Foundation. Dr. Gobbo holds the Special Chair in Interlinguistics and Esperanto at the University of Amsterdam (UvA), Faculty of Humanities, on behalf of the Universal Esperanto Association (UEA, Rotterdam). The content and opinions expressed in this article are the authors′ and they do not necessarily reflect the opinions of the institutions supporting them.

References

Beavers AF (2011) Historicizing floridi: the question of method, the state of the profession, and the timeliness of floridi's philosophy of information. Etica Polit Ethics Polit XIII(2):225–275

Bridges DS, Vîţă LS (2011) Apartness and uniformity: a constructive development. Springer, Berlin

Brooks FP (1995) The mythical man-month: essays on software engineering, Anniversary Edition edn. Addison Wesley Longman, Boston

Burks AW, Goldstine HH, von Neumann J (1946) Preliminary discussion of the logical design of an electronic computing instrument, Institute for Advanced Study, Princeton, New Jersey

Costantini S, Gobbo F (2013) A history of autonomous agents: from thinking machines to machines for thinking. In: Local proceedings of the nature of computation. 9th conference on computability in Europe, CiE 2013. Milan, Italy

Floridi L (2008) The method of levels of abstraction. Minds Mach 18(3):303–329

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

Floridi L (2013) Information quality. Philos Technol 26:1–26

Floridi L (2013) Technology's in-betweeness. Philos Technol 26:111–115

Fortnow L (2013) The golden ticket: P, NP, and the search for the impossible. Princeton University Press, Princeton

Floridi L (2013) The ethics of information. Oxford University Press, Oxford

Gobbo F, Benini M (2013) From ancient to modern computing: a history of information hiding. IEEE Ann Hist Comput 35(3): 33–39. doi:10.1109/MAHC.2013.1

Gobbo F, Benini M (forthcoming) Why zombies can't write significant source code. J Exp Theor Artif Intell

Hennessy JL, Patterson DA (2006) Computer architecture: a quantitative approach, 4th edn. Morgan Kaufmann Publishers, Burlington

Kay AC (1972) A personal computer for children of all ages. In: Proceedings of the ACM annual conference—vol 1, ACM '72

Papadimitriou CH (1994) Computational complexity. Addison Wesley, Boston

Shoham Y (1990) Agent oriented programming. Technical Report STAN-CS-90-1335 Computer Science Department, Stanford University

The Onlife Group (2013) The onlife manifesto. Tech. rep, European Commission

Turner R (2013) Programming languages as technical artifacts. Philos Technol 1–21. doi:10.1007/s13347-012-0098-z

Upton E, Halfacree G (2013) Raspberry Pi user guide. Wiley, Hoboken

von Neumann J (1993) First draft of a report on the EDVAC. IEEE Ann Hist Comput. doi:10.1109/85.238389