The Phenomenology of Programming
Extended Abstract

Author: Suppressed for Blind Review, HaPoP 2018

December 27, 2017

The Phenomena of Computation Let's say that someone spills a stack of dated cards on the floor, and then starts to pick them up in chronological order, grabbing older cards first and then arranging those relative to each other, and then searching through the pile for the more recent cards, and then pausing to sort clusters separately. We would observe different sorting algorithms (Insertion Sort, Selection Sort, Mergesort) in execution.

Let's say that a teenager asks his mother for permission to go a party. She could say "no", she could refer to the answer she already gave to his sister, or she could tell him to ask his father. We would observe passing the answer by value, by reference, or by name, respectively.

What can we take from these instances of human behavior that look like classic programming phenomena? Standard approaches might provide these obvious interpretations:

- Programming techniques are derived from human activity; humans are executing inherent algorithms.
- Human activity mimics programming techniques; we have learned (or, at least, identified) algorithms from computers.

But what we have here is observed experiences that are interesting without reference to their implementing algorithms. Computer scientists may understand this point by analogy to the semantics of a proposition as a truth-value—clearly satisfactory in many contexts and clearly inadequate in daily discourse. What other lens does philosophy offer?

The Phenomenological Explanatory Paradigm We could look to phenomenology, first-person descriptive analysis of things as they appear, based largely on the work of Husserl. He criticizes naturalism, defined by Thompson (2014) as "the belief that science provides the best account of reality." Husserl proposes that describing phenomena requires stripping away the architectonics of hypotheses (the natural world) and the psychology of the scientist, in order to sense the thing itself. The phenomenological reduction that achieves this renunciation of assumptions about the external world manifests in two "moments" (Cogan 2017):

Epoché: Suspension of interpretation, explanation, preconception in favor of experience; a halt to taking the world for granted.

Reduction: Recognition of acceptedness as acceptedness (rather than veracity); not a propositional attitude, but the insight required to accomplish epoché.

This "reset" purges noumena and causal explanations. Perhaps such a reset in the philosophy of computer science would afford more breadth in the perspectives, approaches, and ideas cultivated.

The Possible Defect of the Computational Explanatory Paradigm An analog of Husserl's argument applies to the computational view of the philosophy of computer science. The current paradigm seeks enlightenment about computing through the construction of theories that induce computational (discrete, symbolic, formal) results—indeed, that preclude anything else—and subsequent application of those theories to aspects of human affairs. But the use of formal methods to elicit formal methods is like the use of naturalism to study natural science; like seeking the noumenal on an assumption of noumenality. It is, to use Husserlian terms, a general positing of computation and everything in it as objectively there, an unrecognized captivation (Moran 2008).

In science, experience achieves validity, and knowledge is worked out, through the interplay of experience and thought. “One might describe the phenomenological approach as an iterative process of ontological disclosure in which a world (relevant social practices or involvement whole) and technology (nexus of relevant technologies) are taken as mutually constitutive interpretive contexts in which the one renders the other intelligible—i.e., grounds it as a seemingly meaningful way to be” (Introna 2017). Such analysis applied to computing phenomena may also prove to be stimulating rather than inhibitory.

The Potential Affordances of the Phenomenology of Computing

- Consider the parameter-passing example. A value (answer, or object) can be given directly, or it can be given by the location or residence of that value, or it can be given by a way to compute it. In programming, to pass by value means that the value is safe, will remain the same, etc. To pass by reference means that the subroutine can alter the value, for better or worse. To pass by name means that the subroutine can call upon other functions for evaluation of the parameters subject to the local context. All of this requres division into calling routine and subroutine across an interface strictly governed by exposed parameters. No such division manifests itself clearly in the first-person experience of the family permissions situation. But what does appear is the placing of responsibility. The phenomenon is the choice of sources for formulation and conveyance of a judgment.
- Consider the sorting example. To abstract a description that captures both scenarios, we note that ordered items can be arranged properly by taking either the sorted items or the unsorted items as the base of operations. In the first-person experience, the next placement is driven by either the need of the sorted set or the presentation of the unsorted set. The phenomenon is a choreography of convenience.
- As a phenomenon of data structures smoothly accessible to first-person experience, but not to computation, consider a set of replica household artifacts, perhaps dinner napkins. We can switch between state (1) thinking of them as *class + count*, that is, identical, when there is no reason to distinguish, and state (2), thinking of them as an *array of individuals* when one or more achieve a distinct condition for some reason, for example, needing repair on the hem. To accommodate this in the computational paradigm requires that those two states be rigidly defined with incompatible declarations, any switching requiring total restructing. The pheonemon is painless, effortless, and virtually simultaneous apprehension of two structured perspectives governing identity.

Even logical concepts “must arise out of an ideational intuition founded on certain experiences” (Husserl 1970, pg. 252). All of these phenomena deserve investigation independent of their digital manifestation in programs, which necessarily begs the phenomenological questions in order to conform to a formal system.

Transcendancy (detachment from the world) is what allows us to interrogate what sorts of things we are dealing with, and what allows us to notice those things in the first place. For Husserl, phenomenology is not a method but a commitment. Although Beavers (2002) refers to a “set of processes and procedures,” Cogan rejects such a procedural view (Cogan 2017). In spite of the breach of faith, the purpose here is indeed to recommend this process. The rendering of the algorithm as an imperative structure demonstrates a result from a quasi-phenomenological approach (Hill 2016). Attention to the phenomena of conscious experience reveals the distinction between the algorithm and the Turing Machine. The first-person experience of an algorithm is what to do, an instruction, a command, rather than a color or a shape or a feeling, or even a theorem or formal language. Computational models such as Turing Machines are declarative; they are given by definitions. Algorithms are not. On that analysis, we gain insights into both algorithms and computing.

1. Algorithms are imperative structures.
2. Computation relies on non-declarative objects.

Conclusion On the mild reading, we suggest, as a complement to formal studies, more experiential attention to computing phenomena. On a more pointed reading, we call for respect for computing phenomena as they appear. On the extreme reading, we deny that the computational approach is adequate. We hope the tension proves productive.

References

Beavers, Anthony F. (2002). "Phenomenology and Artificial Intelligence". In: *Metaphilosophy* 33.1/2.

Cogan (2017). "Phenomenological Redution". In: *Internet Encyclopedia of Philosophy*. Date accessed: 20 December 2017. URL: `http://www.iep.utm.edu/phen-red/`.

Hill, Robin K. (2016). "What an Algorithm Is". In: *Philosophy & Technology* 29.1, pp. 35–59. ISSN: 2210-5433. DOI: `doi:10.1007/s13347-014-0184-5`.

Husserl, Edmund (1970). *Logical Investigations*. Trans. by J. N. Findlay. Vol. I. Humanities Press.

Introna, Lucas (2017). "Phenomenological Approaches to Ethics and Information Technology". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Fall 2017. Metaphysics Research Lab, Stanford University. URL: `https://plato.stanford.edu/archives/fall2017/entries/ethics-it-phenomenology/`.

Moran, Dermot (2008). "Husserl's Transcendental Philosophy and the Critique of Naturalism". In: *Continental Philosophy Review* 41.4, pp. 401–425.

Thompson, Evan (2014). "Review of "Phenomenology and Naturalism: Examining the Relationship between Human Experience and Nature"". In: *Notre Dame Philosophical Reviews*.