

The Affordances of Phenomenology for Programming

Robin K. Hill

University of Wyoming, Philosophy Department

23 March 2018

Fourth Conference on the History and Philosophy of
Programming

Outline

- 1 Programs and Us
- 2 Introduction to Phenomenology
- 3 Phenomena of Programs as Problem Solutions
- 4 Phenomena of Problem Solutions as Programs
- 5 Unbegging the Question
- 6 Phenomena of Program Coding
- 7 Phenomena of Program Execution
- 8 Phenomena of Programs as Innate Skill
- 9 Conclusion

People Sorting Records

Picture someone who spills a stack of dated cards on the floor, and then starts to pick them up in chronological order. He grabs older cards first, arranges those correctly relative to each other, and then searches through the pile for the more recent cards, pausing to sort clusters separately...

A computer scientist might observe different sorting algorithms (Insertion Sort, Selection Sort, Mergesort) in execution.

People Passing the Buck

Let's say that a teenager asks his mother for permission to go a party.

She could take an action:

Say "yes" or "no."

Refer to the answer she gave his sister.

Tell him to ask his father.

People Passing the Buck

Let's say that a teenager asks his mother for permission to go a party.

She could take an action:	A computer scientist might observe:
---------------------------	-------------------------------------

Say "yes" or "no."

Passing by value.

Refer to the answer she gave his sister.

Passing by reference.

Tell him to ask his father.

Passing by name.

Computational Interpretations

In other words, we see people doing “the same kind of thing” as programs.

To pursue this down the standard paths:

- * 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 what we have noticed in programs.

But wait! What about the experiences themselves? Isn't there something else?

In what way can we respect that view of programs?

Computational Blinders?

We are missing something. This is the current view.

The things we talk about in computing are subject ONLY to interpretation and analysis in terms of computation.

This is the broader view.

The things we talk about in computing are subject to interpretation and analysis in terms of computation AND in terms of first-person descriptions of experiences.

An Early 20th-Century European Movement



Edmund
Husserl

The turn of the 20th century saw reaction against the prevailing positivism. One was the **phenomenology** of Husserl and others, in which he promoted **first-person descriptive analysis of things as they appear**, free of assumptions about an external world that exists independently of our consciousness.

Phenomenological Reduction

How? The phenomenological reduction gives guidance:

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); the insight required to accomplish epoché.

Terms of Process

Epoché is not a denial of belief (nor an affirmation), but abstention, a *reset* that purges noumena and causal explanations.

And *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.

Husserl uses the term (translated as) “bracketing” to indicate the setting aside of expectations and preconceptions.

Programming Experiences

Such a reset in the philosophy of programming would afford more diversity of perspective, more breadth in the study of programming.

Programmers themselves see certain things going on as they write code to solve problems. Those are first-person experiences of consciousness.

Programmers turn out programs. Those are not first-person experiences of consciousness.

This author must admit that:

- ① She relies on Husserl's interpreters more than on the original text, which is difficult in translation (and presumably also in the native German).
- ② Phenomenology is quite out of fashion.
- ③ For Husserl, phenomenology is not a method but a commitment. Although some commentators [1] refers to a "set of processes and procedures," others reject such a procedural interpretation [2].

Algorithms are Imperatives

Yet we promote the method, and even claim an instance of the success of the phenomenology of programs:

- 1 Programs are constructs of algorithms
- 2 Computational models such as Turing Machines are declarative; they are given by definitions.
- 3 Algorithms are not declarative, but imperative structures, sequences of commands.

The rendering of the algorithm as an abstract imperative structure demonstrates a phenomenological approach [4].

Parameter Passing is Authority Invocation

The
Affordances of
Phenomenology for
Programming

Robin K. Hill

Programs and
Us

Introduction
to Phenomenology

Phenomena of
Programs as
Problem
Solutions

Phenomena of
Problem
Solutions as
Programs

Unbegging the
Question

Phenomena of
Program
Coding

Phenomena of
Program
Execution

Phenomena of
Programs as
Innate Skill

Conclusion

References

Consider the parameter-passing example. A value (a datum, answer, or object) can be given (1) directly; (2) by location; or (3) by deferral to another function... which requires division into calling routine and subroutine.

No such division appears in the first-person experience of the family permissions situation. What does appear is the placing of responsibility, the choice of sources for formulation and conveyance of a judgment.

Phenomenology gives us a new angle.

Sorting is Mixed

Consider the sorting example. People follow the named algorithms in fits and starts, switching among them easily as dictated by the values ready to hand.

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, in a choreography of convenience.

Tracking Replicas

Consider solutions to the problem of setting the dining table, which often involves a set of replica household artifacts, perhaps plates, or chairs, or dinner napkins.

Conceptualizations of dinner plates:

Class + Count: an exemplar and quantity, when there is no reason to distinguish one from another.

Array of Individuals: when one takes on a distinct feature, for example, needing repair.

Tracking Replicas

Consider solutions to the problem of setting the dining table, which often involves a set of replica household artifacts, perhaps plates, or chairs, or dinner napkins.

Conceptualizations of dinner plates:

Class + Count: an exemplar and quantity, when there is no reason to distinguish one from another.

Array of Individuals: when one takes on a distinct feature, for example, needing repair.

The **phenomenon** is fluid, effortless, simultaneous apprehension of two identification perspectives.

Tracking Replicas

Consider solutions to the problem of setting the dining table, which often involves a set of replica household artifacts, perhaps plates, or chairs, or dinner napkins.

Conceptualizations of dinner plates:

Class + Count: an exemplar and quantity, when there is no reason to distinguish one from another.

Array of Individuals: when one takes on a distinct feature, for example, needing repair.

The **phenomenon** is fluid, effortless, simultaneous apprehension of two identification perspectives.

But the **computation** gives rigid and incompatible definitions, any switching requiring total restructuring.

Programming does not capture the phenomenon.

Exercising Epoché

More obvious examples of real-world phenomena subject to implementation as programs that may be a bad fit:

Spreadsheets Bookkeeping with arithmetic? No—lists, grouping, fixed-column display, that is, formatting.

Social Media Free and open presentation under relevance criteria? No—an anarchy of anonymous publication with no accountability.

Computational blinders?

Exercising Epoché

More obvious examples of real-world phenomena subject to implementation as programs that may be a bad fit:

Spreadsheets Bookkeeping with arithmetic? No—lists, grouping, fixed-column display, that is, formatting.

Social Media Free and open presentation under relevance criteria? No—an anarchy of anonymous publication with no accountability.

Computational blinders? If so, the benefits of removal:

Spreadsheets Better business model, higher productivity.

Social Media Abandonment of attempts to fix with more selection criteria.

Assuming Computation

Main Point: All of these phenomena deserve investigation independent of their computational (digital, discrete, symbolic, formal) manifestation in programs.

An analog of Husserl's criticism emerges: The current paradigm seeks enlightenment about computing through the construction of theories that induce exclusively computational results. It begs the question.

The use of formal methods to explore formal methods is, to use Husserlian terms, an unrecognized captivation [8] (that is, a restriction on imagination, hence, understanding).

Affordances

An affordance is “a property of an object or an aspect of the environment, especially relating to its potential utility, which can be inferred from visual or other perceptual signals; (more generally) a quality or utility which is readily apparent or available.” [10].

We seek the *affordances* of programs via phenomenology, *bracketing* computation, to gain a greater understanding of the computer program.

Robin K. Hill

Programs and
Us

Introduction
to Phe-
nomenology

Phenomena of
Programs as
Problem
Solutions

Phenomena of
Problem
Solutions as
Programs

Unbegging the
Question

Phenomena of
Program
Coding

Phenomena of
Program
Execution

Phenomena of
Programs as
Innate Skill

Conclusion

References

What will make computer scientists uncomfortable:

- Untethering from the constraints of logic.
- Forgoing a human-independent system.
- Subjectivity rather than objectivity.
- Repudiation of the hegemony of the Turing Machine.

These are threats to rigor.

Discomforture

What will make computer scientists uncomfortable:

- Untethering from the constraints of logic.
- Forgoing a human-independent system.
- Subjectivity rather than objectivity.
- Repudiation of the hegemony of the Turing Machine.

These are threats to rigor. But formalists understand that begging the question is the worst threat to rigor.

Where Are the Applications?

The
Affordances of
Phenomenology for
Programming

Robin K. Hill

Programs and
Us

Introduction
to Phenomenology

Phenomena of
Programs as
Problem
Solutions

Phenomena of
Problem
Solutions as
Programs

**Unbegging the
Question**

Phenomena of
Program
Coding

Phenomena of
Program
Execution

Phenomena of
Programs as
Innate Skill

Conclusion

References

You may be wondering...

Where are the exercises in phenomenology that Husserl seemed to expect?

Husserl preaches phenomenology...

Where Are the Applications?

You may be wondering...

Where are the exercises in phenomenology that Husserl seemed to expect?

Husserl preaches phenomenology... but doesn't practice it.

Where Are the Applications?

You may be wondering...

Where are the exercises in phenomenology that Husserl seemed to expect?

Husserl preaches phenomenology... but doesn't practice it.

Dreyfus [3] wants to weaponize it.

Where Are the Applications?

You may be wondering...

Where are the exercises in phenomenology that Husserl seemed to expect?

Husserl preaches phenomenology... but doesn't practice it.

Dreyfus [3] wants to weaponize it.

Beavers [1] wants to formalize it.

Where Are the Applications?

You may be wondering...

Where are the exercises in phenomenology that Husserl seemed to expect?

Husserl preaches phenomenology... but doesn't practice it.

Dreyfus [3] wants to weaponize it.

Beavers [1] wants to formalize it.

No one wants to do it.

Where Are the Applications?

You may be wondering...

Where are the exercises in phenomenology that Husserl seemed to expect?

Husserl preaches phenomenology... but doesn't practice it.

Dreyfus [3] wants to weaponize it.

Beavers [1] wants to formalize it.

No one wants to do it.

In the literature, applications are scarce. (See Ihde [5], McPhail [6]).

What is Bracketed?

You may be wondering...

What does reduction bracket, in the case of programs; what are the assumptions that mislead us without epoché?

- ① That the objects of the universe are discrete, and typed.
- ② That the action of the universe is partial-recursive.
- ③ That the interactions between objects and actions of the universe are algorithmic.

What is Bracketed?

You may be wondering...

What does reduction bracket, in the case of programs; what are the assumptions that mislead us without epoché?

- ① That the objects of the universe are discrete, and typed.
- ② That the action of the universe is partial-recursive.
- ③ That the interactions between objects and actions of the universe are algorithmic.

All may be true! We just want to derive them rather than assume them.

Programmers

Now, finally, let's look at the first-person experiences of composing and testing programs, that is, the cognitive phenomenology, not the sensory [7]. (Is there any sensory phenomenology of programming?)

Is the phenomenology of programming computational, that is, does a programmer in action feel like a computer (mechanical)?

Not really.

The First-Programmer Experience

The first-person conscious experience of the design and coding of programs, systems, circuits, or databases is:

the struggle to figure something out, to force well- or ill-defined goals into mechanical procedures on defined inputs

It leads to satisfaction, frustration, fulfillment, exhaustion, accomplishment... but not to an electronic register state.

A programmer feels external to the program, and does not feel like a computer. (The awareness of our own mental computation is part of the experience, as recognized by Aristotle and Husserl.)

Computers

The Affordances of Phenomenology for Programming

Robin K. Hill

Programs and Us

Introduction to Phenomenology

Phenomena of Programs as Problem Solutions

Phenomena of Problem Solutions as Programs

Unbegging the Question

Phenomena of Program Coding

Phenomena of Program Execution

Phenomena of Programs as Innate Skill

Conclusion

References

Can we look at it from the “first-machine” point of view?

That is, what is the experience of the computer?

Should such a phenomenology be available under computationalism (the theory that the brain is a computer)?

The First-Computer Experience

We can't investigate by analogy... we don't know what it's like to be a bat, and we don't know what it's like to be computation incarnate.

We can simulate computation (examined below), but can we inhabit it? Can we find out what it's like via mental exercise? Nagel [9]: No.

The question seems incoherent because the machine has no point of view. However...

A Special Aspect

We do. AND we are computers.

What's special about phenomenology vis-a-vis computing in particular?

- 1 We are the “first persons” (in the first-person descriptions of experience).
- 2 We execute programs.

Some part of our immediate experience IS computational, as an execution trace or simple grasping of the sequence.

It's Personal

The assumption required by computationalism, or any other view that grants us Turing-computability:

Human cognition understands conditional branches to different actions, and understands the assignment of values to variable, inherently.

Computation is accessible to all rational beings (unlike science). So part of the phenomenology of programming is computational.

It's Also Formal

Computing also has a robust non-phenomenological—even anti-phenomenological—theoretical foundation (like science).

This human-independent theoretical foundation was formulated by our rational minds, distilled from our computational understanding.

We can see it, we can do it, and we can frame it objectively as a human-independent system.

Tension or Paradox?

What can we take from this unique positioning?

Tension or Paradox?

What can we take from this unique positioning?

- The phenomenology of programming brackets the science (computation) to reveal aspects of programs.

Tension or Paradox?

What can we take from this unique positioning?

- The phenomenology of programming brackets the science (computation) to reveal aspects of programs.
- The science (computation) is innate in us, independent of theory.

Tension or Paradox?

What can we take from this unique positioning?

- The phenomenology of programming brackets the science (computation) to reveal aspects of programs.
- The science (computation) is innate in us, independent of theory.
- We have formulated an extensive formal theory of the science (computation).

Tension or Paradox?

What can we take from this unique positioning?

- The phenomenology of programming brackets the science (computation) to reveal aspects of programs.
- The science (computation) is innate in us, independent of theory.
- We have formulated an extensive formal theory of the science (computation).

We make a virtue of necessity and declare this a *productive* tension.

The Interesting Thing

So we come back around to such questions as:

How is seeking permission like parameter passing?

How can a recursive definition be implemented in an imperative algorithm?

How can *class+count* be reconciled with *array of individuals*?

How does the spreadsheet invoke formatting?

What's the bridge? That's the interesting question.

Questions (perhaps meta-questions) to pursue:

- How does the phenomenology of computing augment the understanding granted by the computational view, in particular instances or in general?
- How does the embedding of computation into the apparatus of our own first-person experience affect our thinking about this?

Summary

That the phenomenology of computing is not computational is shown by:

Summary

That the phenomenology of computing is not computational is shown by:

- 1 The nature of problem-solving; we do not see that as computation.

Summary

That the phenomenology of computing is not computational is shown by:

- 1 The nature of problem-solving; we do not see that as computation.
- 2 The nature of programs; algorithms are imperative and the structure of data is flexible.

Summary

That the phenomenology of computing is not computational is shown by:

- 1 The nature of problem-solving; we do not see that as computation.
- 2 The nature of programs; algorithms are imperative and the structure of data is flexible.
- 3 There is no first-computer experience, and we cannot fake it.

Summary

That the phenomenology of computing is not computational is shown by:

- 1 The nature of problem-solving; we do not see that as computation.
- 2 The nature of programs; algorithms are imperative and the structure of data is flexible.
- 3 There is no first-computer experience, and we cannot fake it.
- 4 The experience of programming is subjective affect manipulating mechanical reasoning.

Summary

That the phenomenology of computing is not computational is shown by:

- 1 The nature of problem-solving; we do not see that as computation.
- 2 The nature of programs; algorithms are imperative and the structure of data is flexible.
- 3 There is no first-computer experience, and we cannot fake it.
- 4 The experience of programming is subjective affect manipulating mechanical reasoning.
- 5 Yet we are computers, and we have articulated computation as a formal system.

The Affordances








On the mild reading, we suggest, as a matter of interest, more *bracketed* attention to computing phenomena (free of preconceptions).

On a more forceful reading, we call for equal respect for computing phenomena as they appear.

On the strong reading, we deprecate the computational approach as harmful.

In any case, we should explore the affordances of phenomenology as complementary rather than adversarial to the computational paradigm.

References

-  Anthony F. Beavers. "Phenomenology and Artificial Intelligence". In: *Metaphilosophy* 33.1/2 (Jan. 2002).
-  Cogan. *Phenomenological Reduction*. Date accessed: 20 December 2017. 2017. URL: <http://www.iep.utm.edu/phen-red/>.
-  Hubert L. Dreyfus. *What Computers Can't Do*. Revised Edition. Harper Colophon Books, 1979.
-  Robin K. Hill. "What an Algorithm Is". In: *Philosophy & Technology* 29.1 (Mar. 2016). First published online, 11 January 2015, pp. 35–59. ISSN: 2210-5433. DOI: 10.1007/s13347-014-0184-5.
-  Don Ihde. *Consequences of Phenomenology*. State Universit of New York Press, 1986.
-  Jean C. McPhail. "Phenomenology as Philosophy and Method: Applications to Ways of Doing Special Education". In: *Remedial and Special Education* 16.3 (May 1995), pp. 159–165.
-  Michelle Montague. "The Content of Perceptual Experience". In: *The Oxford Handbook of Philosophy of Mind*. Ed. by Brian P. McLaughlin. Oxford: Clarendon Press, 2009.
-  Dermot Moran. "Husserl's Transcendental Philosophy and the Critique of Naturalism". In: *Continental Philosophy Review* 41.4 (Dec. 2008), pp. 401–425.
-  Thomas Nagel. *Mortal Questions*. Cambridge University Press, 1979.
-  Oxford Dictionaries. *Oxford Living Dictionary*. <https://en.oxforddictionaries.com/definition/>. 2017.