Critique of 'Lector in Codigo or The Role of the Reader'

Luke Church
University of Cambridge
Cambridge, United Kingdom
luke@church.name

ABSTRACT

In 'Lector in Codigo', Videla challenges us to write for people and computers simultaneously. I examine the implications of this challenge.

CCS CONCEPTS

• **Human-centered computing** → *Natural language interfaces*; • **Social and professional topics** → *History of programming languages*;

KEYWORDS

literate programming, software engineering

1 CRITIQUE

In 'Lector in Codigo' [4], Videla draws a parallel between the process of writing literary fiction and writing computer programs. Drawing on Eco [2] he introduces the core ideas of the model reader - notional person for whom the author writes, levels of readership - that different readers may read the same text at different levels of aesthetic appreciation, and the encyclopedia as a way of thinking about the knowledge that is shared between the writer and readers of a piece of code.

These analogies ultimately raise the question: for whom do we write? Videla is reminding us that we write programs for other programmers as well as for compilers and imaginary machines. In encouraging us to empathise with, and design systems for, others to read Videla is continuing a tradition of literate programming from Knuth [3], continued in works like NewSpeak [1].

But do these readers, the compiler and the programmer, really form a system of levels, as Eco's readers do?

For readers to form levels there has to be a broad notion of comparison between their needs. Two human readers might well share this, but it's less obvious with modern programming technologies and modern programmers. Imagine giving a draft of a novel to a friend, who refuses to even think about your ideas because on page 235 there was a missing full-stop!

The needs of modern programming technology, as currently conceived, are so extreme in their pursuit of syntactic and type correctness that they distort any conversation with their pedantry.

The style argued for in Lector in Codigo is fundamentally one of unification - that the same text should be interpreted by both readers. In this case the neediest of the readers will always determine the style, and indeed professional programmers have grown accustomed to translating what they wish to say to meet the needs of the compiler and in doing so have forgotten the differences between their readers.

The languages we design have enabled this behaviour. Objects in particular offer a sophisticated form of semiotic deception. The signs in `blindMan.pat(elephant)`, mean very different things to the computer and to a fellow programmer. The computer interprets it as instructions about numbers, but a programmer sees the words as analogies, taken together to form an allegory. The use of the same words for both enables a lazy conflation between whether the signified is a bunch of numbers, or a complex human experience, where any rendering into an object structure will ever be, at best, partial.

In asking 'for whom do we write?', Videla wonderfully highlights these questions - that we currently engage in the pretence that we can have a discussion about the beautiful ideas of programming, by explaining them to each other in a conversation mediated by the mindless pedantry of compilers, IDEs, type checkers and linters.

Surely, there must be a better way?

REFERENCES

[1] Gilad Bracha. 2017. Live Literate Programming. Keynote at <Programming> 2017, Brussels, Belgium.
[2] Umberto Eco. 1979. *Lector in fabula. La cooperazione interpretative nei testi narrativi.* Milan: Bompiani.
[3] Donald Ervin Knuth. 1992. *Literate programming.* Center for the Study of Language and Information Stanford, CA.
[4] Alvaro Videla. 2018. Lector in Codigo or The Role of the Reader. In *Proceedings of the 2nd International Conference on the Art, Science and Engineering of Programming (<Programming'18> Companion).* ACM, New York, NY, USA.

Author version. Suggested citation: Luke Church. 2018. Critique of 'Lector in Codigo or the Role of the Reader'. In Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming (Programming'18 Companion). ACM, New York, NY, USA, 187-187. DOI: https://doi.org/10.1145/3191697.3214327