

in this a4, code is the textual aspect of computer technology that may be loaded up on a text editor and easily changed. i'm consciously avoiding any discussion on the subject of text editors -- write your edits on a piece of paper and then write them to newfile with echo. what's important to me is that this be easily accomplished. with enough time and energy, anyone could write interesting code. the best project would be if my grandmother took the time to re-write the linux kernel from scratch, and if she kept a record of her reflections about code. as much as i like pierre menard, this is not feasible. the processes of code should be manageable without the need of resorting to too much external technical support.

when it comes down to it, this means that code is text written in one of the computer languages. code is the active practice of altering half-understood text files. code are the static characters that silently stare back at you, and that will not even give you the illusion that you're double-guessing a machine. it's always evident that someone else was there before you, and that that person was sloppy. it's just a matter of playing along and locating those three characters in a text file that will make all the difference on whether your computer can display postscript or whether it will keep those secrets to itself. the code is as flawed as the coder.

seldom does the small press have any real political weight. it's very nice when it does, but most of the time it's just poetry by way of fragile, limited, utopian a4s. the small press may be weak, but this is no reason to get rid of it. likewise for code. free software activism may be hot air, but this is no reason to ban floss practices. the code of free software is poetry way beyond high brow computer science. some people have pointed to its infinite reproducibility. other parts of the larger text deal with its glitches.

in order to start coding, consumers don't need to wait for a standards body to define a syntax. if we're already coding, i don't see why we shouldn't start writing our own syntax. this means that there's room for both [1] syntax that's functional from an engineer's point of view [2] syntax that's not functional from an engineer's point of view.

on the one hand, [2] above means code that crashes. on the other hand, it also means that we can start looking for computations outside of computers. on the city like socialfiction did, or at a temp job like that text from the tempslave zine did. this also means that the languages will proliferate. how far will the incompatibilities get?

at the office

in typewriters, printing acts like a screen in that it displays what's typed. unless it's imagined, a typewriter has no random access memory, no disk space, no processor memory. physical traces are left on the typewriter stationary. the erased characters are all there. teletypes received instructions and displayed computations by printing. at the shell, which belongs to the teletype family tree, printing has been abstracted with stuff like lpr. on youtube there's a video where a frustrated office worker starts photocopying the computer's screen directly after the printer sprays him with toner.

could we start coding the office stationary? could we start coding the different paper-printer-staple-computer arrangements in an office or on a desk? calculating their permutations is trivial. how much could we calculate by moving stuff around a desk?

in the user manual of the printer where this was printed, there are the following specifications that may come in handy to the end-user:

model name, print speed, acoustic noise, weight, package weight, external dimensions, emulation, ram, supported sizes of paper, paper smoothness