

code

the essay deals with two things: code, and self-publishing. regarding self-publishing the focus is on zine culture of the 1980s and 1990s. if the the utopian publishing of those years was an afterthought of consumer computing, how does this energy segue back into technology? taking this question as a point of departure, there are roughly three levels to the essay: [1] the essay builds a place where the question may be asked [2] the essay argues that it makes sense to ask it [3] the essay tries to articulate possible answers.

i think of code as consumer text. its purpose is to give us the illusion of agency regarding the technologies upon which we have come to depend. if the internet was invented so that banks could erase their end-user support department, then we can always daydream about the promises of free software.

code are those machinic instructions written in one or another computer language. it can be anything else, as long as it's somehow translatable into that format. likewise, if something is code, then hobbyists like us should be able to deal with it without resorting to too much external technical support.

in this, it is fruitful to remember the shortcomings of the small press. it's naive to put so much faith on it. likewise, it's naive to put so much faith on the possibility of coding. free software activism may be an illusion, but that is no a reason to ban floss practices. there's poetry in the frailty of code, and in its utopian aspects. my code belongs to you and viceversa, code is infinitely reproducible, anyone can code.

zines

there are two things that interest me about the zines of the 1980s and the 1990s. not all utopian publishing is like this, but it's not my intention to define exactly what happened in those publications. it's more like i want to see how these tendencies could evolve:

A zines are media used by their authors to dissent and critique

B zines are media whose design is self-explanatory

A is a rephrasing of underground, and B is a rephrasing of diy. B means that zines wore their design on their jacket. i'm using 'design' in a perhaps idiosyncratic way that includes not only layout but also manufacture and distribution. in their manufacture, there was little or no professionalization, and in their distribution the separation to the audience blurred. the next section goes into more detail about B.

crap hound

both photocopiers and photocopies may be used as publishing tools. more than other media, photocopies invite photocopying. their learning curve is small. xerox collages are self-explanatory, and anyone can do them.

the zine ovo of the late 1980s approached this in an interesting way. it described itself as ``a magazine published on an irregular basis introducing new works into the public domain.'' in this respect, it's now is available not only in pdf format, but also in open office ``source.''

another zine that thought of itself as a tool was crap hound. it was originally published by sean tejaratchi between 1994 and 1998. it was taken up again in 2005. tejaratchi's original intentions were entirely pragmatic. it was to be merely an ``encyclopedia of clipart'':

i was getting paid corporate wages at adidas [...] and i wanted to do something worthwhile with my earnings [...] if i was going to do a zine, i wanted it to be something useful and relevant. i started calling it clip art, but it's changed--my motives are now officially different.

the zine functions as a catalogue of re-usable clip art samples. it's also a collection of collages that sample someone's flootsam of found black and white imagery. it even functioned as a pre-digital distribution medium for fonts:

i've been making fonts for a while, and i've been putting a few in each issue of crap hound [...] [they] are a throwback to dover books. not everyone wants to use a computer. there's nothing wrong with scissors and glue sticks.

manuals

i would hope that all of this would at this point in the essay lead the reader to re-consider what print matter is or could be. for the rest of the essay, print matter is something that manifests itself through or inside of computers. among other things, this means that print matter is no longer fixed. setting something on paper doesn't amount to much.

let's imagine the following situation. there's a hypothetical perl script S that when run prints the following to standard output: ``hello world!'' for me that script is print matter. what i mean to stress with the example is the fluid, variable, unstable form of print matter. print matter is text that at some point in time manifests itself on or through the computer in some way.

this is not as strange as it seems. a similar transformation happens every time a browser renders an html document. under this light, dynamic pages are work that exploits the variable nature of a script's output.

though the script S is probably not the best example, it exemplifies something that is very common. typing something like 'man ls' at the command line is very common. man pages of unix systems are like the script S in that they're both 'dormant' and must be run by their readers.

gnu

in the gnu coding standards, the terms documentation and manual practically mean the same thing. specifically, a manual is a texinfo file, and documentation includes manuals and other things like NEWS files and change logs. in practice, the latter term practically takes the place of the former.

the people of gnu distinguish between the way in which a program was built, and the way in which it's used. they warn about modelling the documentation after the software:

programmers tend to carry over the structure of the program as the structure for its documentation. but this structure is not necessarily good for explaining how to use the program [...] learn to notice when you have unthinkingly structured the documentation like the implementation, stop yourself, and look for better alternatives.

the coding standards advise authors to approach users pedagogically, thinking about ``the concepts and questions that a user will have in mind when reading it.'' what's more, the manuals that they write should admit two types of reading: tutorial and reference. it's interesting to note that by tutorial they mean something that a user may want to read straight through.

regarding unix man pages, gnu suggests that -if adequate- help2man be used to extract a man page from the texinfo file. unlike man pages, gnu manuals should have a ``coherent topic''. as an example, the coding standards note that diff and diff3 are both covered in a single manual, whereas there are two man pages, one for each command.

mythical man-month

according to the mmm, software has two faces, both as important. the code speaks to the machine, and the documentation ``tells its story to the human user.''

in contrast to the approach of gnu, where -in the guise of documentation as tutorial- writers are asked to provide for readers who ``know[s] nothing about the topic,''' brooks assumes that all users are acquainted with the software. for him, there are three types of users:

the casual user of a program, [...] the user who must depend upon a program, and [...] the user who must adapt a program

brooks attributes the poverty of most documentation to the difficulty inherent in trying to keep any two pieces of data in sync. in particular, it will be difficult to keep the ``two faces'' in sync. in light of this he proposes self-documentation:

the solution, i think, is to merge the files, to incorporate the documentation in the source program.

he points to three levels where he could see this happening: [1] ``labels, declaration statements, and symbolic names [...] convey[ing] as much meaning as possible to the reader''; [2] ``use space and format as much as possible to improve readability and show subordination and nesting''; [3] ``insert the necessary prose documentation into the program as paragraphs of comment.''' before proceeding, i should mention that gnu follows brooks in self-documentation. what they deem to be documentation is included in their source code distributions.

non-standard documentation

we could work with the format of software documentation. in brooks' second point, the jump to ascii art is implicit. it could be said that the pre tag goes towards abiding by that point. however, as this well-known work [<http://www.jodi.org/>](http://www.jodi.org/) does we may choose to ignore these code display conventions. in it, the ascii art looks like noise because the whitespace has collapsed.

i like jodi's approach because it uses obscurity at two levels. on the one hand, it looks obscure -- some people mistake their work for a computer failure. on the other hand, it points to how arbitrary and obscure brooks' conventions can be outside of software engineering.