

<!--

Thesis

Code Marginalia – About commenting on web code

Stephan Thiel

Klasse Digitale Grafik

HFBK Hamburg

Bachelor of Fine Arts, 2025

Advisors:

Christoph Knoth

Konrad Renner

Elisa Linseisen

How did I end up here?

```
<!-- I'm thinking about which language to use to
write this whole thing. My literature is mixed,
and my notes are all over the place. I'm asking
friends for opinions on this. I think my English
is okay, but I could probably express myself
better in German. But my gut is telling me, "Oh,
it's web/code related, so I guess English it is."
Let's hold this thought; it will reappear later.
-->
```

```
<!-- I've been thinking a lot about the tone of
this thesis. Thinking back to the last thesis I
wrote, it had to have this scientific, neutral
tone. In my previous thesis, I had to adopt
a tone that was objective, devoid of personal
emotions, and focused on presenting facts and
analysis. With the HFBK and the freedom it offers
to write my thesis, I'm pretty happy, as I want
this to be more personal than I had the chance to
do before. -->
```

How did I end up here?

As always, it all comes back to childhood fears, but let's start with the present. In a way, all of this feels quite Freudian. With me joining Klasse Digitale Grafik, I honestly did not think I'd end up coding. I originally joined because of my interest in graphic design and my general interest in topics related to the Internet. I knew coding was a part of this class, but I never thought I would develop such an interest in the subject of code—primarily because my emotional connection to code used to be one of fear. Over the last few years, I gradually managed to overcome this deeply rooted fear of coding. The more I overcame it, the more fun I started to have while coding my small little websites. With this, which feels like a generic progression for a person learning a new skill, I became more interested in how other websites were made. With this, my descent into the rabbit hole of web programming started to accelerate, and I think I crossed a point of no return with coding.

One of my first PC memories is sitting in front of the family PC running Windows 98, doodling in Paint. I don't have any recollection of surfing the web at this age. This changed when I got my first Laptop in 2002—I was 10 or 11 years old at that point.

<!-- I've just talked to my dad about it—unlike me, he has a good memory. He bought the Laptop on sale at a Real near Berlin in 2002. -->

<a> <https://www.codecademy.com/resources/blog/myspace-and-the-coding-legacy>

¹ Here is a really nice segment about the shift from personalized social media profiles like MySpace to a unified visual appearance. EP 23 Neverpost—Looks Great: The End of Personalization
<https://share.transistor.fm/s/92d2f3fe>

<!-- These are moments where I'm glad I kept all my old school reports. It seems like I finished the course with a grade of 2 (B), which surprised me because I managed to partake in it and get a good grade. Moments like these remind me that my memories are nothing more than canonized fanfiction of myself. Out of interest, I looked at the state of the school's website from when I was attending seventh grade. My old school seemed to go through at least three different domains and had multiple domains running in parallel.

2001–2010 <a> <https://web.archive.org/web/20050210103407/http://www.lilienthal-og.de>

With this, I also got my own connection to the Internet at some point, probably by still plugging a cable from the router into the LAN port of my Laptop. With a personal device and a connection to the Internet, I was well on my way to becoming a very online teen. With the spread of social media and microblogging in the mid-2000s, I had a cringy MySpace page and was later very active on Tumblr. Both sites allow you to customize your page or blog.¹ I really enjoyed all the creativity people had with styling and personalizing their little web spaces, yet I never did that because I was terrified of coding. With MySpace, I'm not so sure how I customized it anymore, but like Tumblr, it also had a code editor. With Tumblr, however, I vividly remember this moment of opening the custom CSS panel many times and closing it again and again in fear. Ultimately, I always used themes—not that this is bad or worse in any way than doing it yourself. However, it had somewhat of a sour aftertaste for me because I always wanted to make my own stuff but could not out of fear and feelings of inadequacy. While researching this thesis and thinking about my relation to code, many memories from my childhood and early teens came back. I remembered why I used to be so afraid of coding—it was the *Informationstechnischer Grundkurs* I had in 2005, which was seventh grade.

2007–2015 <a> <https://web.archive.org/web/20100505023629/http://www.lilicidsnet.de>

2008–today <a> <https://web.archive.org/web/20250114122602/https://lilienthal-gymnasium-berlin.de>

It seems like this <a> <https://web.archive.org/web/20050210103407/http://www.lilienthal-og.de> was the web design that was thought to me at that point.

-->

<a> <https://web.archive.org/web/20040208140124/http://www.lilienthal-og.de:80/seiten/ITG/itg.html>

I remember being taught VBA Basic, HTML with CSS, and using CorelDraw to make graphics for the website. The only other thing I remember is constantly feeling too stupid to program. *I'm not good enough at maths or logical thinking to understand what happens when I type things into the computer.* I was so happy once this course was over, and at that point, it was very clear to me that I was just too stupid to do anything with the web and coding in general. Looking back on this episode of my life and reevaluating this, I can see that is not true at all.

Berlin had a school reform in 2003, which mandated that every school had to have a focus. Our school chose a media focus and had to scramble together an informatics teacher and a course. We were the second year where this course was thought, so everything was still pretty rough. If I remember correctly, the guy who taught us was also the web admin for the school website and not a regular teacher at my school. With me now having been part of a few coding workshops and gaining some teaching experience, I realize that this whole course was not a good way to introduce new students to coding. So, all in all, many different random factors came together. My first year in a new school environment, me being a very angsty and sensitive kid, a school reform that introduced the *Informationstechnischer Grundkurs* the year before, and we were still being guinea pigs for this new curriculum. After this somewhat traumatic first encounter with code, I was convinced I was too stupid to code because it's something only smart or STEM people can do—which sadly is a widespread belief held by people inside and outside the coding field. Although I was socialized as a boy and was a

```
<!-- Sadly, I never discovered net art at that point in time, which probably would have changed a lot for me. -->
```

```
<!-- I still feel stumbling around. -->
```

```
<!-- I am struggling with organizing the next block of ideas. There are many somewhat related things, but I'm wondering what to include. Now, two days later, I think I have worked out everything. I had to rethink the structure of the text, but now I think I'm quite okay with the structure. Thank you to all who supported me emotionally and in terms of content. -->
```

somewhat geeky teen, which was at least stereotypically a prerequisite for becoming part of the coding subculture, I never became a part of it and developed in a different direction.

Besides fear, the bad vibes I felt as a teen were another significant factor in why I never really became part of the male-dominated geek culture. Later, I would learn the concepts to understand what these bad vibes were—sexism, elitism, racism, anti-queerness, toxic masculinity, to name a few. I was mainly on Tumblr but occasionally visited places like *Something Awful* or *4chan* to see what was happening in the abyss. When *Gamergate* happened in 2014, the hate reached my bubble of depressed, internet-obsessed, queer young adults on Tumblr almost instantly and radicalized me in my beliefs. Since then, I had a somewhat split perception of the Internet. I was part of heavily online communities that could make their own Internet to a certain degree, even if that was impossible for me personally.

With the preconception that many people who could make websites or the Internet at large were part of a group of people I was at odds with and my general fear of code, the idea of making the Internet was very far away. Yet, I ended up coding and love making websites.

With my coding skills improving over the last two years I'm now at the point where I have somewhat of a praxis or at least ideas about what I am doing with the medium website, how I am engaging with it, and what I want to do with it moving forward. For me personally, this has had an impact on the way I program, albeit subconsciously until recently. While searching for a topic for my thesis, I began realizing that many of the things I'm drawn to in coding are, in a way, a reconciliation with past fears of my childhood self.

```
<!-- I really love the idea of the site
spirit on polinsski's
https://polinsski.digitale-
grafik.com
current website. It reminds me of Tsukumogami
in Japanese Folklore; they are tools that
have acquired a soul after being used for
100 years and cause mischief if they haven't
been properly cared for. A sort of fictional
anthropomorphization of the website or a form of
world-building or storytelling that happens. -->
```

I like websites with a certain warm feeling—a sort of personal character. This can apply to design and content, as well as to code or how they were made. The aspect of how it's made started to become more pronounced the more I could code. With this, I also realized I wanted the code side of the website to have a warm and inviting feeling, as this was something I had feared and distanced myself from for such a long time.

A feeling that peeking behind the curtain of design is welcome and encouraged. Like the website that invites you in so you can spend some time with it. Or maybe like a connection to the person or people who made the website by encountering artifacts they left behind—by chance or by purpose.

Most websites I visit outside big platforms, like social media or video streaming services, are made by a single person or a tiny group of people. I enjoy getting to know these strangers, who also like making websites just by visiting the spaces they created. So, I want the same to be true when someone visits one of the websites I made. When I leave or encounter messages, notes, explanations, or other artifacts in the code of a website, they are comments. It's one of the few ways to leave text in code that is not code itself. So, over the last few years, the comment grew very close to my heart, and in a way, I'm dedicating this text to you, dear comment. To get to know you better, understand you more, learn about your history, how you're used, and what you mean to me. So I welcome you, dear reader, to go on this journey with me.

What are you, dear comment?

What are you, dear comment?

I like the definition of a comment in Markus Krajewski's text *Kulturtechnik Programmieren*, as it is close to how I feel about it. He talks about the two layers of code. The first layer is the instructions, which are executed by the machine. This is the actual code or text written in a programming language. The second layer is marked text that is not executed. This piece of text is ignored by the machine.² It can be text, code, a mix, or something like ASCII art. Either way, it does not matter to the machine but to us as readers. To make a comment or mark text so it will not be executed, you will have to use a specific string of symbols to let the program know what follows will be a comment. Here is a small example with HTML.³

```
<p>This is a tag used in HTML to define a paragraph of text. What follows below is a comment</p>
```

```
<!-- In HTML, you use this combination of symbols to declare a comment. Everything inside here will not show up when you normally view a website. -->
```

There are two ways to format a comment: block and inline. A block comment is comparable to a paragraph of text with one or multiple line breaks. This would be used for a longer comment. Compared to this, an inline comment is shorter and spans only one line. Here is a quick example of those two types of comments.

```
<p>dear reader</p> <!-- What follows is an inline comment in the same line-->`
```

```
<!-- After this inline comment, I'm currently writing a longer block comment. It spans more than one line of code and has a line break. In the case of HTML, it can be a bit confusing because there is no difference in the symbols used to signify whether it's an inline or a block comment. -->
```

This can be handled differently depending on the programming language. There is a lot of variation in the different symbol strings that mark a comment.⁴

² Bajohr, Hannes, and Markus Krajewski, eds. *Quellcodekritik: zur Philologie von Algorithmen*. Erste Auflage. August Akademie. Berlin: August Verlag, 2024, 66.

³ Hyper Text Markup Language—It's what you use to write a webpage.

⁴ Here is a compilation of how to write a comment in a lot of different programming languages. In: Davids Kacs. "Comments in Different Programming Languages." Accessed April 7, 2025. <https://gist.github.com/dk949/88b2652284234f723decaeb84db2576c>.

```
<!-- Originally, this text was written in Obsidian,
a note-taking software that uses Markdown
  <a> https://daringfireball.net/
    projects/markdown </a>
as its sort of programming language. The comments
I am writing inside Obsidian are declared by
typing %% to open and close it. All the text that
follows will be treated as a comment. -->
```

5 Of course, there are many potential uses of a comment besides communication. One that comes to mind is commenting out code. If the goal is to have a piece of code not perform while still keeping it, the code can be put in a comment. In a way, this code is stored in a state of limbo, hidden from the machine, but maybe also not that relevant to the human reader.

6 In a way, all of this can probably be traced back to the seminal book *The Art of Programming* by Donald Knuth. A lot of the texts I read trace this statement back to Knuth.

7 I really like the wording of performing code instead of executing code here; I took it from Hannes Bajohrs and Markus Krajewski's introduction to their book. In: Bajohr, Hannes, and Markus Krajewski, eds. *Quellcodekritik: zur Philologie von Algorithmen*. Erste Auflage. August Akademie. Berlin: August Verlag, 2024, 15.

8 Chandra, Vikram. *Geek Sublime: The Beauty of Code, the Code of Beauty*. Minneapolis, Minnesota: Graywolf Press, 2014, 14-5.

For the programming languages I'm familiar with they are as follows:

- HTML uses this `<!-- comment -->` for both inline and block comments.
- CSS uses this `/* comment */` also for both inline and block comments.
- JavaScript and php both use `//comment` for inline comments and `/* comment */` for block comments.

Either way, even though the way to mark a text as a comment differs for many programming languages, the functions stay the same. The one I want to focus on is communicating something that is intended for human readers.⁵ As something that seems trivial but has been picked up repeatedly by different authors, the code has a certain duality to it.⁶ It is read by humans as well as machines. More specifically, it is a text read by humans and performed⁷ by machines. This raises the question of who we are writing code for: machine or human? It may seem like we are writing code for the machine to tell it what to do, but are we really? I like how Vikram Chandra puts Donald Knuth's struggle with this question in *Geek Sublime*.

If ever there was a person who fluently spoke the native idiom of machines, it is Knuth, computing's great living sage. More than anyone else, he understands the paradox that programmers write code for other humans, not for machines: "Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do."⁸

<a>
<http://www.literateprogramming.com/index.html>

9 Knuth, D. E. "Literate Programming." *The Computer Journal* 27, no. 2 (February 1, 1984): 97–111. <https://doi.org/10.1093/comjnl/27.2.97>; and Knuth, Donald Ervin. *Literate Programming*. CSLI Lecture Notes, no. 27. Stanford, Calif.: Center for the Study of Language and Information, 1992.

10 Ross Williams. "Literate Programming." Accessed April 7, 2025. <http://www.literateprogramming.com/index.html>.

11 Knuth really committed to literate programming. He created WEB, later *CWEB* <https://www-cs-faculty.stanford.edu/~knuth/cweb.html>, a whole documentation system. Adjacent to this, he also created *TeX* <https://tug.org/whatis.html>, a typesetting system because he was unhappy with digital type settings at that time, which then led him to create MetaFont to make fonts look good in TeX. Two examples of literate programming are Knuth's publications *The TeXbook* and *The METAFONTbook*, both of which used web WEB and TeX in the process.

12 Marino, Mark C. *Critical Code Studies*. Software Studies. Cambridge, Massachusetts: The MIT Press, 2020.

13 Bajohr, Hannes, and Markus Krajewski, eds. *Quellcodekritik: zur Philologie von Algorithmen*. Erste Auflage. August Akademie. Berlin: August Verlag, 2024.

14 Bajohr, Hannes, and Markus Krajewski, eds. *Quellcodekritik: zur Philologie von Algorithmen*. Erste Auflage. August Akademie. Berlin: August Verlag, 2024, 64.

Knuth, a big proponent of writing code meant to be read and understood by humans, coined the term *literate programming* in his 1984 paper and later, in 1992, wrote a book about it. In his case, he advocated for code as a form of discourse between people—the coder as writer and essayist. For Knuth, code should be clear, legible, and communicate its function. Additionally, all the code should be documented in a way that explains how exactly the code works in an elaborate literary way.⁹ He suggests writing code so that big parts of the code are only important to the human reader, not the performing machine. An excerpt from Ross Williams sums this up nicely on Daniel Mall's website literateprogramming.com.

A traditional computer program consists of a text file containing program code. Scattered in amongst the program code are comments which describe the various parts of the code. In literate programming the emphasis is reversed. Instead of writing code containing documentation, the literate programmer writes documentation containing code.¹⁰

The focus is on the human. The comment is one way to achieve this programming paradigm, as it has a singular human audience. Knuth went down his own rabbit hole with this and created a whole system to program this way.¹¹ This also questions the important part of the code—for Knuth, it's the comment or the documentation. Similarly, certain currents in literary studies, such as Critical Code Studies or Source Code Criticism, also question this topic. While Critical Code Studies focus more on the meaning and implications of code in terms of the social context¹², Source Code Criticism is more about developing and generating an understanding of code by enriching the code with knowledge about its origin, function and context.¹³ Bajohr and Krajewski's take on Source Code Criticism is that the comment is a central tool for documenting and explaining code for further study without interrupting the aspect of the machine performing the code.¹⁴ This way of thinking about the relation between code and comment really resonates with me. Going into this text, I had some of my own ideas about the relation between code and comment, and it's always nice to realize there were others before me who also thought about this.

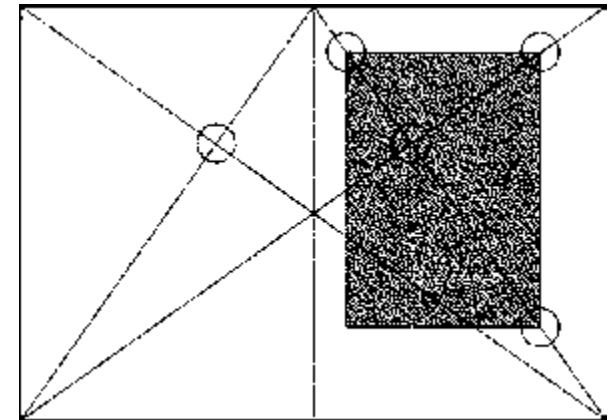
```
<!-- I also get the same feeling when coding.  
There are always people out there who had the  
same problem and already solved it. -->
```

Everything is so relational—Code or writing, it always happens on the back of others who came before me. Quoting other writing, copying code others have written, writing notes, and writing comments. While writing this text, there is a multitude of different annotations, like my personal notes, all the comments, and footnotes. I'm wondering what the important part is—my writing, personal anecdotes, or the original texts of others I refer to in the footnotes, with this text being a kind of fluff piece to contain them. What is the central or essential part of the writing, and what should be highlighted?



Marginalia in a handwritten book from the Middle Ages

Approaching this from the position of a graphic designer, there is also always a decision to be made about the importance of the running text and everything adjacent to it. What are the relations between the font sizes, how much space will I give footnotes on a page, will they be part of the margin or the running text, and how big will I make the margin? Thinking about the aesthetics of the very traditional ways of book setting from the Middle Ages or the more modern formalized constructions of the print area from the likes of Jan Tschichold, they seem to me like there is an emphasis on big margins.



Print area construction for a book page

15 Looking deeper into this relationship between our relationship to books, how we use them, and how margins evolved in modern typesetting or printmaking is a rabbit hole I will not go down, but I'm curious what kind of relationships could be uncovered.

16 As for how to define marginalia, there is the Merriam Webster definition of marginalia: 1. marginal notes or embellishments (*as in a book*) 2. nonessential items. As it is with marginalia, this dual definition also feels like a good fit for comments.

As a side note: I also really like the definition of the english Wikipedia article on marginalia as it includes links to all the different forms of graphical elements (*embellishments in the Merriam Webster definition*) like doodles or drolleries. There is also this really nice Interview from the State Library of Victoria about Marginalia that sums up what marginalia are and also shows some nice examples. <https://www.youtube.com/watch?v=skUs534pBj4>

<a> <https://kimkleinert.com>

<a> <https://polinsski.digitale-grafik.com>

<a> <https://www.are.na/marginalia/channels>

Apart from the arguments on aesthetics, there is plenty of space for notes. To me, these gestures emphasize commenting, taking notes, or writing your own thoughts in a book that has the physical space for it.¹⁵ In short, there is a lot of space for marginalia—all the sorts of things that are happening in the margins of a book.¹⁶ I like that marginalia are a somewhat open space for what they can be. Everything that happens in the margins or outside of the print area is available for anyone—a sort of free commons on the book page. This feeling does not translate one-to-one for me when going from writing to coding—or a text editor to a coding environment. All of these thoughts of footnotes, quotes, annotations, and marginalia meld together into a comment—or more like they all exist in the space between the opening and closing string of a comment. Although I included marginalia into this space, that is, the comment out of all of the forms of different annotations, marginalia feels closest to comments in terms of openness—as comments, as well as marginalia, can be pretty open spaces in terms of what happens inside them. This also brings me to the title of this thesis *Code Marginalia*. For me, marginalia serves primarily as a lens through which to view comments, to figure out what they are, how to approach them, and how to connect them to other forms of annotation. This title and topic are also in relation to the ongoing research Kim Kleinert and Polina Lobanova about Marginalia on the Web.

<!-- Going back to how it came to this specific topic, I think my talks with Polina about websites, making websites, and the poetic web are factors why I decided to write about this topic. I have been interested in comments for a long time. I was also looking at source code to find comments, but talking to Polina before and after their workshop

<a> <https://words-on-margins.website> about the topic of code marginalia was the last push that made me decide to also take a deeper look at comments and what they mean for me. -->

A look into the past

<a> <https://handschriftenportal.de> <a>

17 Screti, Zoe. "Finding the Marginal in Marginalia: The Importance of Including Marginalia Descriptions in Catalog Entries." *Collections: A Journal for Museum and Archives Professionals* 20, no. 1 (March 2024): 122–41. <https://doi.org/10.1177/15501906231220976>, 122.

18 I can really recommend this article as a further read. The main topic of the article is a deeper look into all of the marginalized groups of people that have a voice in the margins, but are ignored by the current practices of digitizing and catalogued books.

To better understand where comments are coming from, I would like to take a closer look at the history of comments and coding in general. But I also want to take a short detour and glance at the history of marginalia.

A look into the past

A very close friend of mine currently works for the State Library of Berlin, where she is working for a project called Handschriftenportal. It's a digital repository of European book manuscripts in German collections, providing a platform for researchers interested in knowledge preserved in medieval and early modern handwritten books. Since then, I've got a lot of pictures of the weirdest (*cute!*) little illustrations that my friend has found over time—they are marginalia. A year ago, I would have never guessed that there would be a connection between my degree and the screenshots of marginalia from the Middle Ages on my phone. We have not talked about this weird coincidence yet, but I asked her if her more research-focused colleagues have some articles on marginalia they could recommend. They recommended a recent article by Zoe Screti *Finding the Marginal in Marginalia*, which advocates for the inclusion of recording descriptions of marginalia in catalog entries to empower relevant yet still marginalized voices in the margins.¹⁷ Screti sums up marginalia's history and recent discourse before advocating for a shift in the current cataloging practice.¹⁸

Screti defines marginalia as follows:

Today, as then, marginalia is used to describe any form of markings (both visual and textual, printed and handwritten) made on a text that are usually—but not always—located in the margins of the said text.¹⁹

The word traces its history back to 1640, when it was used for the first time to refer to printed marginal notes that provided additional context to the main text. But the practice of writing in books is much older than the term—before marginalia, it was *apostille*—deriving from the Latin word *postilla* for note. Screti notes that marking a book is an active response to what a reader has read. In that way, marginalia leaves a trail of personal thoughts and reactions behind, a document of the dialog between the reader, the author, and the text. For researchers, they are an invaluable resource providing glimpses into the readers and their socio-cultural contexts.²⁰ This approach of understanding the socio-cultural contexts through annotations can be traced to the digital with Source Code Criticism, as it employs comparable approaches. Bajohr and Krajewski trace the function of the modern comment through time back to commenting in the legal practice since late antiquity and the theological exegesis—the explanation and interpretation of a text, especially the Bible.²¹ At least for exegesis, this happened—as far as I understood it, in the margins and was also transferred by scribes into later copies of the books. Coming from the digital, it's also interesting to read about the shift to the digital in the field of marginalia research. Screti discusses the digital representation of marginalia, mentioning the slow paradigm shift from using the term marginalia to *paratextual material*, which encompasses the full range of marginal notation.

Screti also refers to Tatiana Nikolaeva Nikolova-Houston's classification of marginalia within hypertext theory and her comparison of the function of marginalia with hyperlinks on websites.²² Even though Nikolova-Houston's view connects more to hyperlinks than comments, it's interesting that there is this overlap in marginalia and HTML. Helen J. Burgess, a media studies researcher, connects marginalia and printmaking to HTML

19 Screti, Zoe. "Finding the Marginal in Marginalia: The Importance of Including Marginalia Descriptions in Catalog Entries." *Collections: A Journal for Museum and Archives Professionals* 20, no. 1 (March 2024): 122–41. <https://doi.org/10.1177/15501906231220976>, 124.

20 Screti, Zoe. "Finding the Marginal in Marginalia: The Importance of Including Marginalia Descriptions in Catalog Entries." *Collections: A Journal for Museum and Archives Professionals* 20, no. 1 (March 2024): 122–41. <https://doi.org/10.1177/15501906231220976>, 122–4.

21 Bajohr, Hannes, and Markus Krajewski, eds. *Quellcodekritik: zur Philologie von Algorithmen*. Erste Auflage. August Akademie. Berlin: August Verlag, 2024, 81-3.

22 Screti, Zoe. "Finding the Marginal in Marginalia: The Importance of Including Marginalia Descriptions in Catalog Entries." *Collections: A Journal for Museum and Archives Professionals* 20, no. 1 (March 2024): 122–41. <https://doi.org/10.1177/15501906231220976>, 124.

<!-- I'm wondering if I should have called my thesis code paratextual materials, this just rolls off the tongue. -->

23 Dilger, Bradley J., and Jeff Rice, eds. *From A to <A>: Keywords of Markup*. Minneapolis: University of Minnesota Press, 2010.

24 The pecia system was a system for efficiently reproducing handwritten in the Middle Ages. Copy texts, so-called exemplars, were loaned out in parts—in piecemeal or Latin pecia, to be copied and then returned. This preserved the quality of the copied books by limiting errors. With this system, multiple workshops worked on one copy of a book, which meant there was a need for communication between all the different people. Pecia marks were the solution to this problem, as they marked where the different pieces of hand-copied text (*pecia*) would be placed in the new copy of the book. In: Dilger, Bradley J., and Jeff Rice, eds. *From A to <A>: Keywords of Markup*. Minneapolis: University of Minnesota Press, 2010, 170–1.

<a> <https://www.metmuseum.org/art/collection/search/466179>

25 Dilger, Bradley J., and Jeff Rice, eds. *From A to <A>: Keywords of Markup*. Minneapolis: University of Minnesota Press, 2010, 184-5

26 Department of Medieval Art and the Cloisters. “The Art of the Book in the Middle Ages,” n.d. http://www.metmuseum.org/toah/hd/book/hd_book.htm.

27 Editorial philology tries to find out more about the origin and structure of texts. Comments clarify ambiguous or unclear passages or information in a text. They highlight deviations or deletions of text between copies and the original to make all the edits transparent. This feels close to commenting out code and writing a note, but let’s be honest, it’s clearly `git` and version control, right?! <https://git-scm.com>

28 Bajohr, Hannes, and Markus Krajewski, eds. *Quellcodekritik: zur Philologie von Algorithmen*. Erste Auflage. August Akademie. Berlin: August Verlag, 2024, 81-3.

and comments.²³ Burgess connects the pecia system, which was used in the process of hand copying medieval books²⁴—and the marks the process leaves as marginalia—to HTML marking sections of content and comments.

For her, comments share formal similarities to pecia marks, as they are both marking sections and spaces for content. Same as Bajohr and Krajewski, Burges draws a parallel between comments and exegesis. For her, these aspects of explanation and interpretation are the key differences to pecia marks. Comments as literary devices seek to explain what will happen when code is performed—they are a companion to code and a tool for communication with other people.²⁵ In this sense, she aligns with the values of *Literate Programming*. All these perspectives reaffirm to me a deep connection between comments and marginalia. On the one hand, both have similar functions in the field of annotations—especially if one considers commentaries as paratextual material, where I understand the textual as code. On the other hand, through the connection between comments, in this case as a form of literary criticism - not code comments, as seen in exegesis and law, marginalia, in which most of them take place,²⁶ and the modern comment - the one in code - which can trace its history back to literary comment. When I look at the relation between this commentary in the margins and the text it is commenting on, it gives me such *Literate Programming* vibes.

Going back to Bajohr and Krajewski and their short summary of the history of the comment, they also shine a light on comments in editorial philology²⁷ to review the comment as a tool for reflection through the lens of Source Code Criticism. They emphasize the cognitive gap between text and comment and the automatic change of perspective one can get from crossing this gap. This continuous change of perspective creates a distance to the subject matter and serves as a means of self-reflection and critical examination of the writing process.²⁸ I think this mechanism also applies outside of Source Code Criticism. I often feel like this while writing comments to myself while coding or, in this case, writing. When coding, I tend to write what I want the function to do as a comment before and iterate between code and text while coding and adding or editing comments and notes. Sometimes, I also write paper notes because I feel that the motion of handwriting helps me

```
<!-- When I look around
both physically and digitally, I'm surrounded
by comments and notes. Currently, I'm writing
at a desk in my living room—it's close to 8pm.
I'm nearly done for today, and I am fried. On
my left is a jumble of notes—random thoughts on
paper while I'm not at my computer; a Kaufland
receipt with notes from one of my flatmates, who
took the time to listen to me while I had my
first hopefully last meltdown about this thesis
and later helped me with structuring; a few mind
maps trying to conjure a semblance of a common
thread through my thoughts into this text; lastly
two Kimchi recipes. In my writing environment, a
few sources are currently open as tabs, and my
random thoughts about the process of writing this
text, how the website and the print version will
look, are going either into this text itself, my
documents called thoughts and design & writing.
-->
```

About your modern incarnation, as the comment in code

more than typing the same thought—or it won't come out unless I'm writing by hand.

About your modern incarnation, as the comment in code

Mixing and matching whatever medium to record a thought or leave a note feels close to how I perceive the transition from handwritten comments for code to digital comments in code. I was vaguely aware of punch cards, but with how coding works now, the early era of computing feels so far away. The first general-purpose electronic computer was the ENIAC. The purpose and associated development of this computer were ballistic calculations for the US military—it was therefore used to calculate the trajectories of artillery and missiles so that they would not miss their targets.

```
<!-- I've been thinking a lot about
how to structure this part—or more like how
visible I want the perceived role of women
in part of history and the origins of modern
computing as a military project. I wrote a few
different versions, mentioning them as a comment
and later a footnote, but I think it's good to
already have this information in the main text—
even though I will have a deeper dive into this
topic in the following chapter. -->
```

29 University of Pennsylvania. “Celebrating Penn Engineering History: ENIAC.” Accessed April 7, 2025. <https://www.seas.upenn.edu/about/history-heritage/eniac>.

30 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 14-5.

31 These two posts about writing code and comments made me daydream about stacks of notes, loose papers, and quick scribbles. One about [comments in early programming](https://www.quora.com/How-long-have-comments-been-able-to-be-used-in-programming-languages-Was-there-ever-a-language-that-did-not-allow-comments) <https://www.quora.com/How-long-have-comments-been-able-to-be-used-in-programming-languages-Was-there-ever-a-language-that-did-not-allow-comments> and the other about [rewriting an old FORTRAN program from punch cards in Visual Basic](https://blog.acthompson.net/2017/05/punch-cards-comments-and-learning-from.html?m=1) <https://blog.acthompson.net/2017/05/punch-cards-comments-and-learning-from.html?m=1>

32 Maybe there will at some point be a time in my life when I visit archives to look at those code marginalia.

Like many technical inventions that we rely on today, the computer too began as a project of the military-industrial complex.

The ENIAC was operated with knobs, plugboards, and an external memory with punch cards. The program had to be written on a piece of paper and then translated into binary for the machine to understand and input into the different parts of the computer by turning the right knobs, plugging cables into the plugboards, and ingesting punch cards.²⁹ These crucial and complicated steps for operating this computer were performed by women—called the ENIAC Girls, who were forgotten in history for a long time. They were mostly regarded as nothing more than operators—clerks akin to switchboard operators—by their male colleagues.³⁰

In this era of handwritten or typewritten code, I wonder how many notes and comments were written down on paper—part code, part notes, and scribbles. I imagine little reminders and drawings for remembering the specific quirks of this machine, notes to self, and notes to others. Scribbles, crossed-out errors, or rewritten letters and numbers because they were not readable enough. Notes to help with figuring out how a specific program should work and notes for the conversion to binary. Notes and markings on specific punch cards and maybe even comments on the final versions of binary instructions.³¹ All of this happens in the margins of the instructions fed to ENIAC to perform code. I think that was a time when comments could be considered marginalia in the classical sense.³²

Coding was very tedious and challenging in this era of computing. Over the coming decades, new approaches and ways of programming were found step by step, which also transformed the form of comment. With the change from binary machine language to the first level of abstraction assembly language in the 1950s and the subsequent

```
<!-- It was so hard to grasp how early computing
with knobs, switchboards, and punchcards worked.
In the beginning, I had no idea how code would
be transferred to something like a punchcard or
tape because all the literature I read assumed
knowledge about this process. So, I turned to
Science YouTube for help and was rewarded. I'm
such a sucker for good science communication.
After binging a lot of different YouTube videos
and taking notes, I'm glad I finally have a rough
overview of the history of computing - even if
it's just through the lens of comments. At the
beginning of my research, I focused more on
the social aspect of coding. Still, I realized
that I really have to get to know the basics of
the technical aspect of these early computing
technologies to understand how this transition
from paper to digital came and how comments
transitioned in this period. -->
```

33 Chandra, Vikram. *Geek Sublime: The Beauty of Code, the Code of Beauty*. Minneapolis, Minnesota: Graywolf Press, 2014, 40–1.

34 Chandra, Vikram. *Geek Sublime: The Beauty of Code, the Code of Beauty*. Minneapolis, Minnesota: Graywolf Press, 2014, 92.

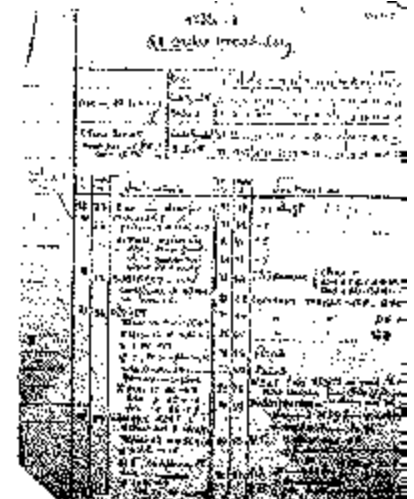
35 These two videos from PBS Crash Course summarize the early history of computers as well as programming languages. CrashCourse, “Early Programming: Crash Course Computer Science #10,” May 3, 2017, accessed April 7, 2025, <https://www.youtube.com/watch?v=nwDq4adJwzM>; and CrashCourse, “The First Programming Languages: Crash Course Computer Science #11,” May 10, 2017, accessed April 7, 2025, <https://www.youtube.com/watch?v=RU1u-js7db8>.

change from low-level programming languages to high-level programming languages such as FORTRAN³³ and COBOL³⁴ in the 1960s—the comment stayed a companion.

I present to you a brief summary of the history of computers from the perspective of the comments. Throughout researching this transition period³⁵ and getting to know all these different technologies, I've noticed that even though the hardware changes over time, the programming languages and their features are always the ones that are supported on the devices. So, the focus through all these iterations of computers and storage media lies on the language of code and the human element, not necessarily the device. Nevertheless, the devices are still important in understanding this transformation process, as these devices determine the medium on which writing takes place—initially analog and later digital. In the 1950s—the era of ENIAC and its successor devices—computers used different media for storing and loading programs or code. These were knobs and switches, plugboards with cables, paper tape as well as punch cards.

The basic workflow from coder to computer was the following:

- 1 Write down your basic idea of what the code should do by hand.
- 2 Next, you write the actual code on a coding form. A piece of paper that is comparable to a code editor today. The code would have been in Assembly or, later, a high-level programming language. If you write code for the ENIAC, you will have to translate the code into Machine Language by hand.



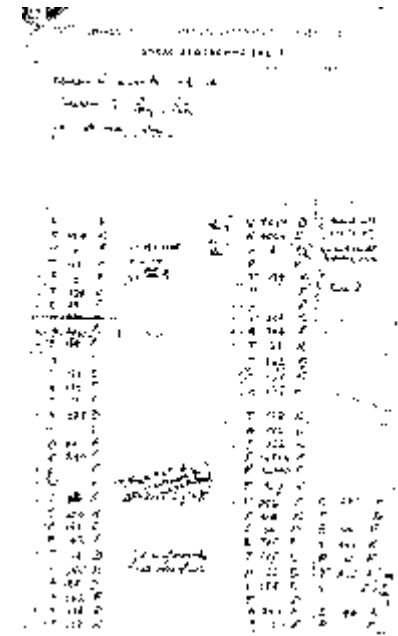
ENIAC Program Development Sheet

ADDRESS	OPERATION	OPERANDS	REMARKS
0000	START		
0001	LOAD	DATA	LOAD REGISTER 2
0002	ADD	10	ADD 10
0003	STORE	DATA	THIS HAS EFFECT OF MULTIPLYING BY 2
0004	LOAD	DATA	WORK RELATIVE ADDRESSING
0005	ADD	DATA	
0006	STORE	DATA	
0007	LOAD	DATA	CONVERT TO DECIMAL
0008	END		END OF JOB
0009			
0010			
0011			
0012			
0013			
0014			
0015			
0016			
0017			
0018			
0019			
0020			
0021			
0022			
0023			
0024			
0025			
0026			
0027			
0028			
0029			
0030			
0031			
0032			
0033			
0034			
0035			
0036			
0037			
0038			
0039			
0040			
0041			
0042			
0043			
0044			
0045			
0046			
0047			
0048			
0049			
0050			
0051			
0052			
0053			
0054			
0055			
0056			
0057			
0058			
0059			
0060			
0061			
0062			
0063			
0064			
0065			
0066			
0067			
0068			
0069			
0070			
0071			
0072			
0073			
0074			
0075			
0076			
0077			
0078			
0079			
0080			
0081			
0082			
0083			
0084			
0085			
0086			
0087			
0088			
0089			
0090			
0091			
0092			
0093			
0094			
0095			
0096			
0097			
0098			
0099			

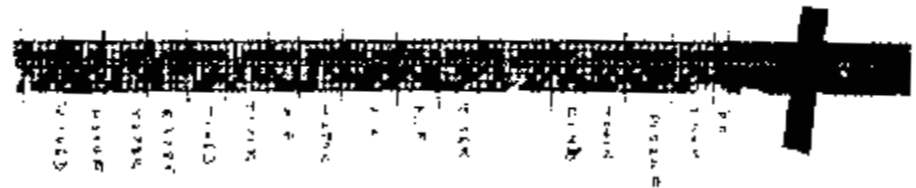
Assembler Code Form

3 Next, you transfer your code to a memory medium the computer accepts. These could have been a series of knobs, where you have to switch them on and represent the binary code. Another way would have been a plugboard, where you connect outputs to each other with cables. The third way was punch cards or paper tape. With them, you could type your code into a dedicated machine that would punch the paper holes accordingly. In the case of the punch card, one card represents one line of code.

4 Run the program.



EDSAC programme sheet



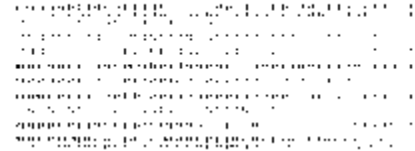
and corresponding paper tape instruction strip

This process did not change much between the different programming languages. While writing code, there was always space for comments, notes, or scribbles on the paper—it did not depend on the language, and all code was prewritten on coding forms. The early high-level languages like FORTRAN³⁶ and COBOL³⁷ had the option to also write comments directly in the code by marking the line with the right symbol. These could also be encoded onto the punch cards, with the additional option of commenting out any

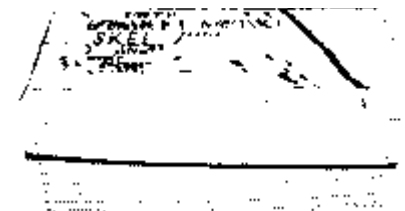
36 GNU. "28.14.3 Fortran Comments." Accessed April 7, 2025. https://www.gnu.org/software/emacs/manual/html_node/emacs/Fortran-Comments.html.

37 mainframestechhelp. "Cobol Tutorial—Comment Line." Accessed April 7, 2025. https://www.mainframestechhelp.com/tutorials/cobol/comment-line.htm#google_vignette.

line of code afterward by punching out a certain field on the card. The whole punch card would be treated as a comment—irrelevant to the machine but still an integral part of the order of the punch card stack. In a way, the comment could be transferred from the margins of the coding form to a new state. It would be embedded in the code's main text but still retain its status as an outsider, as it would not be seen as relevant for part of the code's readership - the machine.³⁸



A punch card is one line of code



Software A stack of punch cards

38 With this in mind, I'm wondering if everything that was printed on a punch card—be it the numbering, logos, or, in general, anything that was not the punched hole, could be considered something akin to comment or marginalia from the perspective of the machine.

39 Éric Lévénéz. "Computer Languages History." Accessed April 7, 2025. <https://www.levenez.com/lang>.

40 Davids Kacs. "Comments in Different Programming Languages." Accessed April 7, 2025. <https://gist.github.com/dk949/88b2652284234f723decaeb84db2576c>.

41 CrashCourse, "Operating Systems: Crash Course Computer Science #18," June 28, 2017, accessed April 7, 2025, <https://www.youtube.com/watch?v=RU1u-js7db8>.

42 Multics. "Multics History." Accessed April 7, 2025. <https://www.multicians.org/general.html>.

43 Multics. "Multics Features." Accessed April 7, 2025. <https://www.multicians.org/features.html>.

44 CrashCourse, "Operating Systems: Crash Course Computer Science #18," June 28, 2017, accessed April 7, 2025, <https://www.youtube.com/watch?v=RU1u-js7db8>.

With most of the following high-level programming languages evolving from those FORTRAN and COBOL,³⁹ the function to comment in the code itself also was part of everything that followed.⁴⁰ The next era of computers would move one from punch cards to computers with actual operating systems.

The hardware changed, but the possibility of commenting on the code remained. You were able to comment directly in the terminal on the computer itself. With these new computers and their operating systems, multiple people could share one computer through many terminals—single access points with a screen and a keyboard—and use them to run their programs.⁴¹ One example of these early operating systems is Multics.⁴² The operating system could run programs directly from a terminal in many programming languages.⁴³ Later, with the development of Operating Systems like Unix, the computer retained most of the functions, but the computer and the terminal were the same. So, a computer with one user—a personal computer—brings us to the contemporary comment.⁴⁴

<!-- When I read Miriam Humm's thesis on Hyper-text while researching my own, I found out about the Advent of Computing Podcast, which she linked as a playlist. I have listened to many different episodes since then and can really recommend it if you are interested in these topics after reading about them here.

Here is a Playlist of all the Episodes that are interesting for the topics this text touches on.

ENIAC-Part I <a> <https://adventofcomputing.libsyn.com/episode-43-eniac-part-i>

ENIAC-Part II <a> <https://adventofcomputing.libsyn.com/episode-44-eniac-part-ii>

Assembly <a> <https://adventofcomputing.libsyn.com/episode-140-assembling-code>

FORTRAN <a> <https://adventofcomputing.libsyn.com/episode-23-fortran-compilers-and-early-programming>

COBOL <a> <https://adventofcomputing.libsyn.com/episode-60-cobol-never-dies>

C Part I <a> <https://adventofcomputing.libsyn.com/episode-53-c-level-part-i>

C Part II <a> <https://adventofcomputing.libsyn.com/episode-54-c-level-part-ii>

Story of Mel <a> <https://adventofcomputing.libsyn.com/reading-the-story-of-mel>

Esoteric Languages <a> <https://adventofcomputing.libsyn.com/episode-78-intercal-and-esoterica> <a>

What was also really helpful in understanding the history of early computing was this [video series about Computer Science](#)

<a> <https://www.youtube.com/watch?v=tpIctyqH29Q&list=PL8dPuuaLjXtNlUrzyH5r6jN9u1IgzBpdo>

by PBS Crash Course on YouTube. A lot is also more about programming and many other topics, but a few videos touch on the history and basics of many aspects of the technology we use today. programming and many other topics, but a few videos touch on the history and basics of many aspects of the technology we use today. -->

A search for bad vibes

A search for bad vibes

However, this digression into the history of the comment and, hence, also into the history of programming has largely ignored a crucial aspect. The consideration of the social aspect and the social practices that have developed over the decades—a history of what I perceived as bad vibes. And also a history linked to why the comment is so important to me. One of my main sources for understanding the social aspects of the last 80 years of computing history is Nathan Ensmenger's book *The Computer Boys Take Over*, in which he examines the rise of the computer expert in American Society over the last century. However, the questionable history of technology as a male narrative goes back much further.⁴⁵

Going back to the setting of the 1940s, when computing was getting started, women were mostly relegated to clerical positions like human data processing or execution, with men being in charge of the research and development of this new technology. It wasn't any different with the ENIAC. The women who worked on this project set up the machines and should simply implement the instructions of the male planners. They were at the bottom of the hierarchy of intellectual and professional status. Yet the process of translating the instructions into machine code

⁴⁵ Later, after reading Judy Wajcman's *TechnoCapitalism Meets TechnoFeminism*, I learned about Ruth Oldenziel's *Making Technology Masculine*, which presents the historical process of transforming technology as something inherently masculine during the transition from the 19th to the 20th century.

and inputting them into the machine turned out to be not a simple operation. It required innovative thinking and new approaches developed by the female coders. What was thought to be a straightforward process of coding an algorithm turned out to be a multi-layered process of analysis, planning, testing, and troubleshooting. They had to fight for the acknowledgment of their insights and improvements in computing. The so-called *ENIAC Girls* were unknown and not recognized for their contributions for a long time.⁴⁶ These and later many other women in similar positions were a crucial part of this part of computing history, with many becoming programmers. This left the programming field in a somewhat uncertain position regarding the social status connected to the works. Both men, who came from high-status research jobs, and women from low-status clerical positions became programmers. At the same time, the field began the professionalization process to become a high-status discipline. Until the 50s and 60s, the programming profession was still relatively open to women compared to other occupations in the same period, but the advancing professionalization changed this.⁴⁷ This professionalization is a central aspect of the masculinization of programming. Ensmenger refers to Margaret Rossiter and others who have suggested that the process of professionalization is deeply rooted in the exclusion of women.

To improve the status of the discipline, the often feminized aspects of low-status work had to be excluded from the profession. In addition, a prerequisite level of higher education makes access to the profession more difficult. This made it disproportionately hard for women to enter the profession—especially when also encumbered by the role of mother and or wife.⁴⁸ Professionalism also implies a level of authority and expertise—qualities that haven't been associated with femininity.⁴⁹ Ensmenger attributes pressure for increasing professionalization to the following key reasons. The software industry of the 1960s was in great need of programmers as the computerization of business was rapidly expanding. With a shortage of programmers, they started to become highly sought-after workers with an increased status in their jobs.⁵⁰ This, however, collided with the managerial class of workers, as they felt threatened in their positions. In the eyes of managers and office workers, unruly and unpredictable programmers had to struggle to find their place in the company hierarchy.⁵¹ The way I read it,

46 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 14-5.

47 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 243.

48 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 239.

49 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 239.

50 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 236-40.

51 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 22.

professionalizing programming was a way of standardizing this new group of workers. To make them more predictable and cost-efficient and to integrate them fully into the established system without endangering the existing corporate structures.

One way companies tried to select the people for the job of programmer was by testing programs for new employees. These programs often included pseudoscientific aptitude tests and personality profiles. They included a focus on mathematic skills or a formal education in mathematics, which was even disproven when these tests first appeared and favored antisocial men with an interest in mathematics—the detached male programmer.⁵² The goal of those testing programs for companies was to transition from dependence on the few highly skilled programmers to a larger, less skilled workforce, where each programmer is interchangeable.⁵³ With this systematic preselection for a particular type of man to be a programmer, the stereotype of the antisocial, male, mathematics-inclined programmer came to be in the 1960s and was embraced by men as a self-ascription of the profession.⁵⁴ With the increasing masculinization of the field, behavior patterns took root, whose impact I can still feel today. The idea of real programmers who work on real computers with genuine programming languages. They understand the computer and think in machine code. They don't need easy programming languages because they are just that good with code. No one else can probably understand their code because it's just too clever and exploits the hardware on a fundamental level.⁵⁵ The rudeness and antisocial behavior are what makes them a good programmer—no bullshit, just objectively superior code. And in comparison to men, who are born programmers with their logical thinking and problem-solving nature, women just can't compete because they are out of their element.⁵⁶ Glory to the male genius!

52 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 77-9.

53 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 81.

54 Ensmenger, Nathan. *The Computer Boys Take over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge (Mass.): MIT Press, 2010, 239-4.

55 Chandra, Vikram. *Geek Sublime: The Beauty of Code, the Code of Beauty*. Minneapolis, Minnesota: Graywolf Press, 2014, 45-6.

<!-- Just rereading and rewriting these passages from Chandra's and Marino's texts makes me so mad. And it's not only coding; it's everywhere. Okay, I think I'm going for a short walk to calm down a bit. It's so infuriating. -->

56 Chandra, Vikram. *Geek Sublime: The Beauty of Code, the Code of Beauty*. Minneapolis, Minnesota: Graywolf Press, 2014, 58-9.

Marino sees this mix of machoism, sexism, and chauvinism in coding as forms of *encoded chauvinism*. He defines this as the following:

(...) encoded chauvinism, the name I give denigrating expressions of superiority in matters concerning programming, which I see as a foundational element of the toxic climate in programming culture, a climate which often proves hostile—particularly to women and other minority groups—and is a kind of technological imperialism.⁵⁷

This hierarchical view based on an arbitrary judgment of what is *real* or *good* or *right* code can already be seen in the concept of higher- and lower-level programming languages as it implies an order with the machine at the bottom and the human and their language at the top.⁵⁸ It's represented in the age-old discussions of which programming language is the *best* and only *real* programmers using specific languages,⁵⁹ the formation of certain programmer archetypes⁶⁰, and superiority complexes in the open-source community⁶¹—even though this could in theory be a place with good vibes.

All of this builds a narrative about who can code and who should have access to it. Returning to the first comment in this text, where I talked about my struggle to decide on a language to write this text in, the language of code is another substantial factor in the imperialistic—and to a certain degree colonizing—aspects engrained into coding. There is no real alternative to the English language.

Until I got into coding, I was still pretty naive about this, if I'm honest. I remember one of my first thoughts about this was: *How do people who don't speak English or use a keyboard with a Latin alphabet actually code? Yeah, you will probably just use your specific character set, and it will work somehow.* This idea of universal language and character support was shattered as I realized how people in our class would switch their keyboard from their native layout, like Hangul, to English. I realized quickly that there is basically only English in the commercialized world of coding. There are, in comparison to the totality of all programming languages, a few that are non-English. Still, many of them are either not really compatible, proof of concepts, or for educational purposes.⁶² Besides these very few non-English

<!--
There is also this lighthearted [thread](#) about non-English devs talking about switching from their native language to English for coding.

 https://softwareengineering.stackexchange.com/questions/1483/do-people-in-non-english-speaking-countries-code-in-english -->

63 Mateas, Michael, and Nick Montfort. "A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics." In *Proceedings of the 6th Digital Arts and Culture Conference*, 144–53. IT University of Copenhagen, 2005, 148.

64 HTML, CSS, JavaScript and php.

65 Marino, Mark C. *Critical Code Studies*. Software Studies. Cambridge, Massachusetts: The MIT Press, 2020, 131-2.

66 For example, in Donna Haraway's analysis of the connections between technology, capitalism, and patriarchy, and Silvia Federici's analysis of the emergence of capitalism from a feminist-Marxist perspective.

programming languages, the only other exceptions are *esoteric languages*. They are more akin to artistic endeavors probing the boundaries of what a programming language can be.⁶³ The programming languages I and many others use to make websites are only available in English.⁶⁴

This bias is also prevalent in the input devices we use. Even my German qwertz layout is not really that nice for coding. Many of the characters I have to use are hidden behind different combinations of control, shift, and option, whereas they are way more accessible on an English qwerty layout. Plenty of people I know switched permanently to the English layout because of coding.

As Marino puts it, the choice to base programming languages on one specific natural language has significant implications for everyone interacting with it. It can have colonizing effects on non-native speakers, and every programmer who is not a native speaker is at a disadvantage in the computational economy, as they don't have the same connection to the different tokens or syntax adopted from the natural language.⁶⁵ Code and programming are not neutral or unbiased—they are an interwoven mess of the social conditions they emerged from, their integration into the capitalist system and patriarchal structures, characterized by displacement, exploitation, demarcation, and the continuous development in these conditions.⁶⁶ They were and still are, to a certain degree, tools of Western imperialism and US soft power. Nevertheless, they are also tools for expression that can be subverted, rethought, and critically engaged with. Practices like code work, code poetics, electronic literature, or new artistically inclined programming languages broaden our understanding of what code can be and who a programmer is. Similarly, the field of software studies is critically engaging with what code was, is, and can be.

67 LIVINGINTERNET. "Internet History -- One Page Summary." Accessed April 7, 2025. https://www.livinginternet.com/i/ii_summary.htm.

68 Cern. "A Short History of the Web." Accessed April 7, 2025. <https://home.cern/science/computing/birth-web/short-history-web>.

```
<!-- The 1980s were also when HTML
came to be, but this is another story Miriam Humm
already told. <a> https://linking-nodes.net </a>
-->
```

69 Becky Robinson. "Women Invented the Internet, Too," *Medium* (blog), May 13, 2019. <https://medium.com/@rshrobinson/women-invented-the-internet-too-4d3a2fef14ff>.

```
<a> https://cyberfeminismindex.com </a>
```

70 World Wide Web Consortium. "A Little History of the World Wide Web." Accessed April 7, 2025. <https://www.w3.org/History.html>.

71 Hanna, Barbara E., and Juliana De Nooy. *Learning Language and Culture via Public Internet Discussion Forums*. London: Palgrave Macmillan UK, 2009. <https://doi.org/10.1057/9780230235823>, 21; and Wajcman, Judy. "TechnoCapitalism Meets TechnoFeminism: Women and Technology in a Wireless World." *Labour & Industry: A Journal of the Social and Economic Relations of Work* 16, no. 3 (April 2006): 7–20. <https://doi.org/10.1080/10301763.2006.10669327>, 12.

72 Consalvo, M. (2003). Cyberfeminism. In *Encyclopedia of new media* (pp. 108-109). SAGE Publications, Inc., <https://doi.org/10.4135/9781412950657>.

73 Rhizome. "Net Art Anthology." Accessed April 7, 2025. <https://anthology.rhizome.org>.

```
<a> https://are.na/kristoffer-tjalve/poetic-web-calendar </a>
```

74 O'Sullivan, James, and Dene Grigar. "The Origins of Electronic Literature as Net/Web Art." *The SAGE Handbook of Web History*, 2019. <https://api.semanticscholar.org/CorpusID:181858184>, 433-5.

75 I'm positioning my understanding of the current web in line with the theoretical frameworks of platform capitalism, surveillance capitalism, and digital colonialism.

With the web, I see history repeating itself to a certain degree. The technical foundation for the Internet was built as a US military project for distributed communication in the 1960s called ARPANET. First, in the US, later in Europe and other countries, similar networks were established between universities for research purposes.⁶⁷ The World Wide Web, or the Internet as we know it today, got its start in 1989 at CERN⁶⁸; yet again, it is an overwhelmingly male story.⁶⁹ From the original proposal, the web didn't take long to take off.⁷⁰

Cyberspace promised to be full of new possibilities—potentially for everyone. The utopian cyberfeminist visions of completely redefining yourself and leaving markers like race, gender, and class behind stayed a vision⁷¹. Yet again, the web was positioned as part of masculine culture through its closeness to technology and computer science. However, positions such as cyberfeminism have been criticizing and working against these circumstances since the beginning⁷², as have art movements such as net-art⁷³ and electronic literature⁷⁴, which also overlap with cyberfeminism to some extent and also break open what the web, websites, and web-code can be. Parallel to this early period, characterized by curiosity and hobbyism, the web was beginning to be monetized and was fully integrated into capitalism by the 2000s and the bursting of the dot-com bubble. In its current form, the Internet largely exists as an entry point for walled gardens run by big tech fighting for our precious attention and data, as well as the technical backbone of the platform economy at large to slowly commoditize every aspect of our existence.⁷⁵ Somehow, I still love the Internet, at least when I look at the communities existing in resistance to those forces building a web that is not following the path of monetization, like the poetic web for instance.

But what about the comment? On the web, you can see the source code of a website quite easily compared to other software, like the browser itself. The inspector in modern browsers and the view source function display a website's HTML code and provide access to the CSS and JavaScript files a website uses. With this, I can look through the web code and all the comments left behind. With so much of the code behind software hidden or

76 Thompson, Clive. *Coders: The Making of a New Tribe and the Remaking of the World*. Erscheinungsort nicht ermittelbar: Penguin Publishing Group, 2019, 48-9.

77 Jay. "Checking 'Under the Hood' of Code." *The History of the Web* (blog), May 21, 2021. <https://thehistory-oftheweb.com/checking-under-the-hood-of-code>.

<!-- Finding

out more about View Source was a fascinating rabbit hole. I spent half of the four hours trying to write the correct query and browsing random forum posts until I finally found this Quora post from Glen Murphy,

<https://www.quora.com/Who-came-up-with-view-source-in-browsers>

which had somewhat of an answer. The basis of the quoted Blogpost by Berners-Lee testing different Browsers in 1992

<https://www.w3.org/History/19921103-hypertext/hypertext/Viola/Review.html>

I think ViolaWWW was the first browser to support this in 1992. Sadly, I couldn't find this feature mentioned in the documentation I found for ViolaWWW.

<https://web.archive.org/web/20010802155531/http://www.xcf.berkeley.edu/~wei/viola/book/preface.html>

Another contender could be Netscape in 1994. This release history for the browser

https://en.wikipedia.org/wiki/NCSA_Mosaic#NCSA_Mosaic_for_X#NCSA_Mosaic_for_Windows

lists version 2.0 alpha 3, April 6th, 1994, with

inaccessible, I wondered why every modern browser would happily show me everything it could. This can be traced back to the early 1990s and early browser development when developers realized that this could be a fun feature to let people see the website's code as it was sent to the browser either way to render the website. With view source, every website became an opportunity to experiment and learn from—web code became accessible and easily sharable.⁷⁶ In the mid-2000s, the modern integrated web inspector was created. There were different tools with similar functions as stand-alone applications available; building on that, a Firefox plug-in Firebug and the Web Inspector for Safari unified all of the different aspects in 2006 into tools that worked inside the browser. Firebug focused more on debugging JavaScript, while Web Inspector, on the other hand, prioritized viewing source features more. Over time, all browsers adopted the inspector with more or less the same features.⁷⁷

the feature view document source code. The book **Coders: The Making of a New Tribe and the Remaking of the World** by Clive Thompson attributes the view source feature to Netscape. But it's not stated if it was the first browser. -->

78 MySpace 2003, Facebook 2004, Youtube 2005, Twitter 2006, Tumblr 2007, Instagram 2010

79 Dilger, Bradley J., and Jeff Rice, eds. *From A to < A >: Keywords of Markup*. Minneapolis: University of Minnesota Press, 2010, 184-5.

80 Mattias Tornqvist. "The Evolution of JavaScript Frameworks: From jQuery to React and Beyond" *Medium* (blog), June 20, 2023. <https://medium.com/@mattias.trnqvist/the-evolution-of-javascript-frameworks-from-jquery-to-react-and-beyond-f94b34e-7dae8>.

81 John Hughes. "A Brief History of Website Builders," December 2, 2023. <https://wpshout.com/history-of-website-builders/#gref>.

<!-- I want to elaborate more on my position on frameworks, especially template builders. With frameworks targeted at web developers, I think it's great that there are tools that make web development easier or, in many cases, web app development. At the same time, I'm thinking about the reasons for streamlining these processes, and I believe that, in many cases, it can be summarized as optimizing because of market pressure. If the trade is sacrificing the understandability and accessibility of code and, by proxy, reducing the potential agency of people interacting with a given website for economic benefit, I really think it's a bad trade. A trade all of us have to make, but nevertheless, a trade we are somewhat

I think the rise of the inspector and JavaScript debugging tools coincides with the rise of the platforms⁷⁸ and web2.0 in general, with many websites focussing on user-generated content and the websites growing ever more complex. For a big part of the web, you can still peak at what is happening, but you don't get the full picture anymore. As Burgess explains with php. The browser still displays the HTML as the end result, but nobody can see the code for everything that happens on the server, like fetching specific data from a database to build the HTML.⁷⁹ With ever more users on websites like Facebook, Twitter, or Youtube, a growing tech stack to manage and optimize these platforms, and progressing monetization, these websites transform into closed-down applications running on the web. This is also the time of a paradigm shift for the web at large. It moves away from an ethos of sharing code to build your own web to one of proprietary code only seen because of the technological remnants of early web history. For me, the rise of JavaScript frameworks like React or Vue since the 2010s⁸⁰, as well as the increase of template builder websites⁸¹, are symptomatic of this shift, as they streamline the process of making websites and web apps but sacrifice the accessibility and understandability of the code while visiting the website.

forced to make due to our economic circumstances or market pressure in general.

Website template builders, which are used by regular people, designers, and developers, are good things by themselves. The ability of tools to assist people who are not able to code when making websites is wonderful. I think that creating websites and web-based art is not inherently connected to the ability to code, and websites or web-based art is not in any way worse than if it had been coded. At the same time, I think the ability to code opens other avenues to understand the medium of websites, which may be specific to code and are worth exploring. My main issue with the current system and specifically the business model of template builders is the shift from tool to closed ecosystem service. If I can use a website builder as a tool, that's great! If it locks me into a system with limited options and a monthly subscription fee, where I cannot leave without losing my website—not great. On top of that, the generated HTML code has the same issue as the code generated by frameworks, as the potential agency is turned into higher profit margins.

Edit: I could not find a good spot to discuss code minification, so it's also here.

Similar to frameworks, code minification is also a way to optimize web code. It compresses the code by removing all *unnecessary* characters. This loads the JavaScript faster but makes the code unreadable unless it is somehow reworked. Even in that case, all original variable names, comments, etc., are wholly lost. When thinking about optimization in this case, I wonder for whom something is optimized? Certainly not you and me.

-->

For me, apps of websites on the phone and mobile browsers represent the endpoint of this obfuscation process. Apps transform the original websites into fully closed pieces of software, removing them entirely from any ethos of interacting with the code, while mobile browsers take away all options for viewing source code and interacting

with code in general. Reading about trend forecasts predicting that the majority of people connecting to the Internet already experience it exclusively via their mobile devices⁸², I really wonder if the idea of making a website something that is not mediated through a service is already a thing of the past which only a few lost souls, i.e., web designers cling on—so me and maybe you.

This leaves me in a somewhat awkward spot.
For whom am I making websites, and why?
Am I a service as well?

I know why I make websites—it’s fun, and I really enjoy every aspect of making them. The whole process is very personal, which is important to acknowledge. Most personal or professional websites I have made have been designed and programmed by me alone. The few times I collaborated with others, the process was either split between design and programming, or it was close to two people designing and coding on one device. I like to work on things where the project scope stays somewhat small and, to a certain degree, personal. I like collaborating with others, but I don’t enjoy being part of a bigger system where I can only do specific tasks regarding design or programming. I want to have a bond with the things I make and spend so much time with.

For me, this applies to both personal and professional projects. When being compensated for making a website, I’d rather like the website to be a record of a collaboration, a bond, than a service I provide. I want to move away from the notion of the clean, effortless final product that appears out of thin air to be consumed. I want my websites to be imbued with their history—to be both a completed object and the process itself. I realized that this dual nature is why I like the medium of the website so much. Be it on the Internet or even as a local HTML file.

82: GSMA. “From ‘Mobile Only’ Internet to Content Strategies: New GSMA Study Identifies the ‘Megatrends’ Shaping Mobile Industry,” September 11, 2018. <https://www.gsma.com/newsroom/press-release/from-mobile-only-internet-to-content-strategies-new-gsma-study-identifies-the-megatrends-shaping-mobile-industry>.

<!--

This is the first time I’ve experienced this with a medium. As I did a lot of 3D and regular graphic design before making websites, I was always surrounded by digital files. But so much was unseen, with a disconnect between the final object and the process of making it. Having 100 GB of files as the process was reduced to a single JPEG object somehow always felt weird. -->

For me, this duality of object and process translates to the split between the website that exists due to the code being performed and the code that is performed for the website to exist. Switching between these two very different layers and experiences opens up somewhat of a place for me. I can experience the same thing in two different Modi, with potentially very different types of material, without them interfering negatively. If I'm interested in taking a peek behind the curtain, I'm allowed to—I love that curiosity is rewarded.

I want to set up a place for people who engage with the things I make.

I want to set up a place for people who are interested in websites.

I want to set up a place for people who are curious.

I want to set up a place where my past self would have felt welcome.

I want to set up a place where I treat you, me, and web code with care.

There are places like this on the Internet, and I want to contribute to them.

How do you fit into all of this, dear comment?

How do you fit into all of this, dear comment?

The comment, I think, is one way that helps set up places like these. It is a tool for communication, and I want to embrace this. Be it to write to myself, collaborators, or others who see the website. From what I experienced while browsing and writing comments, they can take many different shapes. I'll try to organize my impressions into a few aspects to better describe what I see in these different types of comments. However, I don't understand them as selective categories; rather, they are overlapping fields where a comment can have one or multiple aspects.

The first and most common type is the aspect of explanation. This type of comment is all about describing the code's functionality in a short, clear, and impersonal manner. From my experience, this is also what is used most by the coding community and is generally understood as a comment. When scrolling through Reddit and other forums, as well as skimming over good practice advice, this is what I see most often as the canonized comment, with many debates over what a good or a bad comment is. I imagine scenarios where code is commented on in preparation for a hand-off between developers or while trying to organize the contributions of many people more efficiently. I somewhat see Knuth's ideal of literate programming in this aspect, as it also focuses on explaining code. To a certain degree, it's good practice to describe your code with comments. However, it's something that programmers seem to do afterward and to a lesser degree than they would like to in theory.⁸³ This also explains to me why the more elaborate and work-intensive ways of commenting, like literate programming, never caught on. The way current large language models⁸⁴ comment code also falls for me in this aspect, as they can only reproduce the status quo. Acknowledging this status quo, I think it's good that it is now possible to make written code more understandable by including automatically generated documentation. Looking back at the history of code and computing, this way of engaging with the comment and code in general stays in the confines of the masculine-coded programming tradition. Even with an outlier like literate programming, this aspect has an underlying motive of efficiency and rationality. Code has to be commented on to streamline the working processes. I see this aspect in a continuous struggle to find a balance between writing and not writing comments to maximize efficiency and, by proxy, profit. Further

83 Ljung, Kevin, and Javier Gonzalez-Huerta. "To Clean Code or Not to Clean Code? A Survey Among Practitioners." In *Product-Focused Software Process Improvement*, edited by Davide Taibi, Marco Kuhrmann, Tommi Mikkonen, Jil Klünder, and Pekka Abrahamsson, 13709:298–315. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022. https://doi.org/10.1007/978-3-031-21388-5_21.

84 I deliberately use a large language model instead of artificial intelligence, as this is the underlying technology and not a marketing term used to imply intelligence where there is none.

automating this time-consuming process of commenting in a standardized way using large language models seems to solve this struggle in the near future.

Initially, I still had a way of thinking about comments and code that tended to label them as good and bad to a certain extent. The more I looked into the topic, the more I realized that the debate and evaluation between good and bad is not particularly helpful. It's so shaped by historically grown values that classification is not practical for me, as nearly everything that is important to me in comments exists in opposition to efficiency, productivity, and impersonality.

Re-examining the aspect of explanation outside the bounds of efficiency, the key function of explaining code stays the same, but the goal can change. A website can become a repository of knowledge and a learning tool for others that is more accessible and understandable than just code. In that way, I resonate with Knuth's idea of literate programming, but more in the sense of taking the time to think and reflect about how to communicate what code does in a way that prioritizes the human reading it. I also see the opportunity for citation or referring to others to build a network of references and relations to other people and websites, embedding it into a contemporary period of web-making.

For me, the aspect of the archive represents the contextualization of present and unused code, as well as existing, unutilized, or even rejected features. So, writing about everything that is present, commented out, or not in the code anymore. Reflecting on decisions, recoding processes, documenting features that never came to be, mourning parts that no longer exist, treasuring drafts, and imagining alternative paths. I think of this aspect as a personal interpretation of version control on the website itself.

Moving beyond code itself, I want to talk more about the more personal aspects of commenting—the aspects of relation and intimacy. The aspect of relation represents comments that establish connections between me and something or someone else and position my work in larger or specific contexts that are important to me. I see them as ways to express inspiration, write about references, refer to ideas, recommend something, or share a fascination. The aspect of intimacy is all about emotions, thoughts, and opinions. Reminiscing about moments of joy, voicing

struggles, hardships, and frustrations, telling an in-joke or sharing a secret, holding on to fleeting thoughts, and sharing some gossip. They are the records of day-to-day life while working on a website, little artifacts of moods—a sort of diary.

The aspect of worldbuilding represents narrative elements and storytelling. I consider this aspect, to a certain extent, as an inversion of the aspect of intimacy, as it builds a fictional character that takes the place of the person making the website. Depictions of this could be websites as entities or characters serving as the main focal point of personal interaction and relation. I imagine narratives unfolding as a website grows older, gets updated, or certain milestones are achieved. Mixing storytelling and role play with the making, maintaining, and updating a website.

Last, I want to talk about the aspect of ornamentation, which embodies the urge to decorate and organize. Even though it is not a comment, I would include the general shape of the web code in this aspect. For example, the stylistic choices made for indenting, line width, breaks or white space, and layout in general. This can also include considerations about the web code's general structure. Are different parts of code treated like chapters, or are certain parts of the CSS and JavaScript split between multiple files for specific purposes? Returning to the comment itself, I see it as a blank canvas for custom lettering, headlines, spacers, separation lines, illustrations, and many more forms of ASCII art—playing with the textual nature of code and subverting it. I see the aspect of ornamentation as an extension of the website's design permeating through all layers of the website, as well as a graphical layer that can be used to deepen other aspects.

All of these aspects exist outside the scope of writing functional code for a website to be performed. They are bridges enabling connections to other practices, which are able to enrich the experience of creating web code and interacting with environments of web code. I see comments as ways to relate to and reflect on the code I write. With my now better understanding of the history of code, I also feel the responsibility to critically engage with this part of my craft in an ongoing way—and why not visibly in the documents and websites I create. For me, a part of this is making my code understandable and relatable to

non-programmers as well. In this, I align with many aspects of Bajohr and Krajewski's concept of Source code Criticism from the perspective of a person who writes code. Code in itself should be something that should be primarily written with human readers in mind. In source code criticism, the focus is more on analyzing algorithms and more complex code than on the websites I make. Nevertheless, I think that the approach of contextualizing web code as it is created and placing our actions in a larger context also makes sense for websites. On the one hand, with a view to all the people who visit the website in the present; on the other hand, with a view to the future and the possibility that the website could still be available in some form in archives or databases many years from now. When I search on the WaybackMachine myself, I wish I could find something additional on many websites that would tell me more about the context and the moment in which the website exists or about the people who made it. In the research for this thesis, I could only learn more about the background by reading interviews or memoirs; I wish this had also been possible on the websites themselves.

<a>
<https://rhizome.org/editorial/artbase-anthologies-002>

Recently, I started watching Cory Archangel's [Let's Play Majerus G3](#). It's part artwork and research into the late painter Michel Majerus' restored laptop. Majerus used his computer in nearly all aspects of his working process as it was an extension of his studio into the digital. Archangel shares insights about Majerus's practice by exploring this digital working environment and explores how artists' use of digital tools could have shaped their work. Following this deep dive into Majerus' works left me wondering about artists or designers working exclusively with digital media or net art websites. Relating my own practice to this, I'm wondering how much my laptop would say and how much is already inside the website—as I see it as a working environment supported by my code editor of choice. Using the analogy of the studio to reflect on the dual nature of object and process in websites. Here, the website as the object would be the artwork, and the website as the process would be the studio. In this sense, it feels weird to me that I am artificially cleaning my studio afterward or trying to use my studio space without leaving any traces behind. I see it as a somewhat magical property of websites that I can leave my whole studio within them. I'm also realizing now why I think of websites in terms of spaces or a place.

<!-- This last paragraph started as a comment, but I think it works better in the main text, even if it's a bit scatterbrained? But just now, I was thinking about male living spaces and how they are often somewhat sad or awful places to exist in. Scrolling through [r/malelivingspace](https://www.reddit.com/r/malelivingspace) is a very sobering experience for me, with only a few outliers. When looking at these sad, for me, very cis-male-coded living spaces, I kind of get the same vibes as when looking at desolate web code. -->

⁸⁵I'm talking about, let's call them [Web1.0 aesthetics](https://www.cameronsworld.net) like the websites hosted on [Geocities](https://www.oocities.org/#gsc.tab=0) or revival and nostalgia-driven projects like [Neocities](https://neocities.org/browse)

Using the metaphor of studio, I now start to understand why it's also so important to me to make these spaces nice and warm, as they are places I inhabit or inhabited. Looking at my physical desk right now, I'm in front of my laptop, surrounded by my notes, different pens, snacks, tea, and plants, while my diffuser spreads bergamot fragrance. Of course, this also carries over into my digital working environments.

But building these personal spaces and sharing them with others somehow exudes an aura of unprofessionalism. I associate this link between personal web space and this perceived lack of professionalism with the commercialization of the web in the 2000s and the entry into the era of Web 2.0. The personal, quirky, the undesigned, and the hobbyist had to go⁸⁵ and make way for a cleaner and better-designed look. For me, this professionalism expresses itself in terms of writing web code in code that is clean, efficient, and detached from the person writing it, as it is soaked in the conventions that established themselves throughout the history of coding. There is no room for the playful, the expressive, the opinionated, the relational, the personal. I see it as a way to erase the people working on web code, as well as their spaces. It hides the diverse voices, perspectives, and histories behind a veil of professionalism, further upholding the problematic web-making values.

I think it's important to rethink what professionalism means and how this social code shapes what we do. It's not that I want to go back to yellow Comic Sans on a light green background, but I think it's important to reconsider where the web came from, where it will go, and what influence we have. The current values of professionalism regarding web code and the histories of code and web are

deeply entangled in capitalist and patriarchal structures. As a result, I see how we tend to make the web as fundamentally in conflict with the effort and care it takes to set up these places through comments. In this sense, commenting could also be understood as a form of care within the framework of technofeminist theory. Commenting as a form of work within programming that is geared towards enabling access and participation for others—to care for those visiting and to make those who make the web more visible. I see them as an opportunity to reflect on and change how and for whom we write web code. More than just displaying the code of a website through view source, comments can make this part of a website into a more inviting space—working against the prejudice of who should have access to these spaces. Similar to feminist hackerspaces, I see the opportunity for websites to be places of learning, questioning, and sharing what it means to engage with the technologies we use.⁸⁶ In this sense, comments can create spaces of critique, rethinking, and resistance - not as a singular solution but as one of many possibilities to critically engage with the historically grown structures of our practice as web makers.

*So we need you, dear comment, to carry them—
as web code is not yet able to carry these criti-
cisms within itself.*

*We need you to go to the margins.
We need you to step outside while staying in.
To distance ourselves and change our perspective.
Not because we are pushed aside—but to
embrace an outsider's position.*

*We need you to set up spaces from within in
which we can exist, act, and push for change.
Spaces that we want to inhabit. In which we also
can learn, share, voice our struggles, or have fun.
To gather in them and connect with others—as
so many have before us through time.*

86 Krasny, Elke, Sophie Lingg, Lena Fritsch, Birgit Bosold, and Vera Hofmann, eds. *Radicalizing Care: Feminist and Queer Activism in Curating*. Series of the Academy of Fine Arts Vienna 26. London: Sternberg Press, 2021, 195-7.

