On the subjective nature of programmatic code

Camilla Scholz
Interface Cultures, University Of Arts Linz
Linz, Austria
camilla.scholz@kunstuni-linz.at

Abstract

Bias in technology is rooted deeper than just subjective data. Many sources of bias can be found in its history and creation, thus in the code itself. The culture surrounding Computer Science has historically strengthened discriminatory values, being mostly male-driven and inaccessible. At the same time, through its now ubiquitous and mathematical nature, technology has gained an objective, nearly god-like presence thus making it a powerful influence on the world. This paper challenges the seeming objectivity of code by examining the culture and values surrounding source code as the foundation of technology and autonomous actor in society. By translating goals and actions into source code, meaning is added depending on the context, making it subjective in nature. The context surrounding programming languages and programmatic codes decides what is programmable and how. This interplay is explored through a new esoteric programming language *feel.exe* that utilises emotional expression as code, serving as an artistic intervention that highlights the subjective and personal nature of code and the tension between code and its real-life influence.

Keywords

Esoteric Programming Language, Discrimination, Bias, Value-Encoding, Source Code

Introduction

The topic of discrimination and bias has been frequently discussed in recent years. These discussions, however, focus mainly on the data that is used in, for example, the training of Artificial Intelligence. However, these problems are rooted deeper than just the data. While data does show how machine learning can replicate existing biases, the underlying problem is the culture and social structures surrounding these practices. Ari Schlesinger writes in her thesis“ Addressing Computing’s Discrimination Problem: A Framework for Anti-Discriminatory Computing” that CS (Computer Science) students are “being taught data structures and algorithms with problem sets and programming assignments that [have] implicit social and cultural foci, yet the social and cultural aspects of computing [are] conspicuously absent from the curriculum” (Schlesinger, 2023, pp. 15-16). The definition of what Computer Science entails is often held to be rather narrow, excluding its social responsibilities and thus limiting who it is for.

The particularity of Computer Science is that it appears to be built on the rationality and objectivity of mathematics. However, unlike mathematics, it is applied directly in the real world and should therefore take social aspects into account as well. CS researchers must be aware of their position and see that CS, like any other science, can never be neutral or objective.

Here, a special responsibility lies within the software or source code that technology is built upon. As Bhattarai writes in “Algorithmic Value: Cultural Encoding, Textuality, and the Myth of 'Source Code’”, source code is always executed within a social system and is thus constrained by the system (Bhattarai, 2017, p.4). Source code makes up the interface between the “objective” machine and the non-deterministic human, thus executing the biases in the world.

This paper will analyse this tension, between the seemingly objective or mathematical nature and the social and cultural implications of Computer Science, with a special focus on the meaning of (source) code as the material of technology. First, different standpoints on the entanglement of technology in society are explored, analysing how social and cultural rules influence the creation and application of technology with the example of emotion recognition technology. Then, code and programming languages are critically examined in their role within the discriminatory aspects of technology. Lastly, the paper will reflect on these topics through an artwork. The artwork *feel.exe* is based on a new esoteric programming language that uses emotional expression as code to explore the false objectivity of code through an artistic lens.

The entanglement of society and Computer Science

Not unlike other sciences, the knowledge created in Computer Science and the culture surrounding it influence each other. Knowledge can never be objective but is influenced by the researcher or programmer and the social systems that surround them (Schlesinger, 2023, p.16). Combining this with the often limited notion of Computer Science’s responsibilities, one can observe that people are not often seen as central to CS (Schlesinger, 2023, p.36).

At the same time, programmatic code and technology in general have a seeming objectivity and rationalism to them that resulted from their strong mathematical influence. This image of objectivity can falsely validate the results of CS, making it seem objective and thus increasing the power algorithms have over us. The new media researcher Wendy Chun writes that we tend to equate data with information, focusing more on collecting as much data as we can without questioning how this data is embedded in its surroundings (Chun, 2008, p.315). Thus, it becomes easier to see data and algorithms as the direct source of meaning.

This can lead to problems. Chun argues that we have created a kind of fetish and worship surrounding Computer Science and software specifically. As the network of new technology grows, understanding technology as a whole becomes impossible, thus creating a god-like overarching presence (Chun, 2008, p.300). Applications of technology and algorithms pervade nearly every aspect of our lives, creating a new kind of “enlightenment” with software being the new truth and essence of media (Chun, 2008, p.301).

Returning to the mathematical influences of programming, one can observe a problem in Computer Science’s tendency to reduce reality, especially people, down to easily formattable, mathematical data (Schlesinger, 2023, p.19). This oversimplification paints programmers as magicians and masterminds, simplifying the world, finding out the ultimate truth, while applying these inaccurate models directly to the world. Combining this with technologies' godlike

presence, it becomes clearer why biases are often not as obvious, being hidden behind the image of Computer Science. Chun writes that "We 'primitive folk 'worship source code as a magical entity - as a source of causality - when in truth the power lies elsewhere, most importantly in social and machinic relations" (Chun, 2008, p.311). Finally, as algorithms often operate autonomously, determining who is responsible for their biases becomes increasingly complex.

An Example: The discriminatory history of emotion recognition algorithms

One can observe this disconnect between the values and goals that influenced the creation of technology and its application using the example of emotion recognition software.

The research on affective computing at the MIT Media Lab that resulted in one of the most used emotion recognition software, Affdex, started by using research on autism as a "disorder of empathy" as a justification for this technology. Autistic people were framed as "autistic idiot savants" and as being similar to emotionally unintelligent computers (Nagy, 2024, p.3990). According to Jeff Nagy, this assistive but also dehumanising pretext built on false autism research in emotion recognition AI still influences its use nowadays, although its application is now more widespread.

In this example, one can observe an oversimplification and abstraction of humans into data that strengthens problematic values. With emotion recognition software now often being used for surveillance capitalism, the reduction of the user to a manipulable actor shows how early biases against its initial target group still influence its application nowadays. People are not seen as independent but as teachable, "minable data traces" (Nagy, 2024, p.4000).

This example shows how technology and CS are often strongly influenced by social systems and their values, in this case, a derogatory view of autistic people. As the Associate Professor at Virginia Tech, Ashley Shew writes in "How to get a story wrong: Technoableism, Simulation, and Cyborg Resistance", Computer Science often builds on false narratives when talking about disability or minorities. Disability is seen as brokenness that can be fixed through technology. Ashley Shew describes different tropes of how disability is represented in tech (Shew, 2022, p.14). The example of emotion recognition technology fits the inspirational overcomers trope, where autistic children could overcome their autism with this emotional hearing aid. At the same time, it turns disabled people into objects and programmable data, devaluing the disabled experience.

This example highlights how source code is not separate from its surroundings. The culture surrounding autism research and, thus, CS research influences the code and algorithms, resulting in problems as it clashes with the perceived objectivity of algorithms.

Critical reflection on programming language

The base of all this technology is software and code. Chun describes it as a'" visible invisible 'essence", the "Invisible whole that generates the sensuous parts" (Chun, 2008, p.300). Goals get translated into code to then act in the world. However, translating goals into code, even if it seems direct, introduces biases and its own value systems even before it is executed. Chun writes that when code is read, it already creates an outcome even if it is not executed (Chun, 2008, p.313). Translating a goal or idea into software code adds new meaning to it. Similar to data and information, we tend to equate the written code with the action (Chun, 2008, p.303). However, as soon as code is written and readable, it already creates something in the mind of the reader, creating a gap between the code and its action. The design of the code changes how it is written, and the programming language decides what is more programmable.

As Lucas D. Introna writes in "The Enframing of Code: Agency, Originality and the Plagiarist", the medium with which we create changes what we create (Introna, 2011, p.126). In this translation process, action and agency shift and are always influenced by their surroundings. For example, with the introduction of electronic writing, knowledge could be more organised and new ideas were formed more easily. When this was extended by electronic writing with non-linear editing and copy and paste, the concept of the author changed as one could reuse parts of other texts (Introna, 2011, p.127). The equipment, therefore, changes how we think about the encodings of language.

In the case of programming languages, the social structures surrounding Computer Science have a large influence on what a programming language is and what its main goals are. Programming languages are seen as better when fewer people have the skill to learn them, meaning when they are more inaccessible and stereotypically masculine (Hermans & Schlesinger, 2024, p.212). The Computer Science culture shapes what knowledge is created. The knowledge is shaped by its context. In the context of programming language design, this means that the design reflects the limited diversity of Computer Science culture. Additionally, the context decides what is seen as programmable. For example, the digits 0-9 reflect only a Western perspective of numbers, while, for example, in Arabic, different numerals are used (Hermans & Schlesinger, 2024, p.208). Thus, even just the characters that are written have an influence on what is and can be created and encoded.

Combined with the tendency to abstract people into data points, this leads to not everyone or everything fitting the data requirements. By generalising and abstracting, often from a Western and male perspective, the concept of otherness is created, which includes everything that is not or cannot be encoded. Here, Lucas D. Introna gives the example of emotions or feelings having to be reduced to the possible encoding (Introna, 2011, p.119), which can be observed in the example of emotion recognition algorithms in the previous subchapter. Human behaviour and the emotional reaction of autistic people were categorised into right and wrong,

sorting the autistic experience into otherness, as not programmable. This influence reaches further than the concrete application, as algorithms are used more widely and within contexts they were not intended for in the first place.

Programmatic encodings, similar to legal or moral codes, are rule-based enactments with specific constraints. Source code is not contextless. By applying it to the real world, it is influenced by the systems and organisations of our society. Bhattarai argues that the real problem is not the"source code" or data, but the underlying social and political structures (Bhattarai, 2017, p.2). Thus, a tension between the belief in source codes 'objectivity and the reality of its subjective influences arises.However, unlike legal or moral codes that have to be enacted by humans, programmatic code can act autonomously through machines in the world. This can lead to problems when the code depends and is influenced largely by its context.

Bhattarai argues for open-endedness in algorithms, meaning the "source code" is not deterministic and can have changing results based on the context. Especially in data-driven algorithms, the results change based on the specific en- and decoding of information (Bhattarai, 2017, p.20).

Chun sees code rather as a re-source than a source (Chun, 2008, p.321). Code can only create an action after it is executed, and the execution depends on how the code is compiled or interpreted. Thus, it creates a gap between the code as a source and the result of the execution, making the code into a re-source for action. This translation process is, however, not stable and can change over time depending on its context (Chun, 2008, p.322).

Felienne Hermans writes in "A Case for Feminism in Programming Language Design" that by looking at the programming language community through a feminist or artistic lens, one can explore new pathways in the design and use of programming languages. Not all programming languages need to be hard to learn, inaccessible or written in English. Projects like an Arabic programming language or a new programming language, Hedy by Felienne Hermans, that simplifies textual programming for children, show different approaches (Hermans & Schlesinger, 2024, pp. 215-216) going against the stereotypical CS cultures. Thus, they can question the values embedded in code, opening up what goals and values can or should be important.

Esoteric programming languages

This feminist approach to programming language can break open these constraints.

For example, esoteric programming languages defy the traditional value system of programming languages, playing with what is programmable. An esoteric programming language, or Esolang, is "a computer programming language designed to experiment with weird ideas, to be hard to program in, or as a joke, rather than for practical use" (Esoteric Programming Language, 2024). Esolangs question what a valid programming language is and what it should be created for.

Coming back to the earlier discussion of how code changes within different contexts and the inherent subjectivity of the source code text itself, in particular, multicoding languages reflect this as the code itself and the output both hold meaning within different contexts (Temkin, D., 2020, p.1). For example, to write a program in *Chef*, one writes a cooking recipe as the source code to be executed.
This plurality reflects Chuns' ideas of how source code already has an outcome when it is being read or written without ever having to be executed. This example illustrates how code can be a re-source, not a source of action. The code can be executed in different ways, creating different outputs depending on how it is executed.

An Artistic Intervention : feel.exe

The topics surrounding the subjectivity of code and the meaningfulness of source code in itself influenced the creation of my artwork *feel.exe*. *feel.exe* is an interactive artwork in the form of a new esoteric multicoding programming language where one writes code by writing about one's feelings, thus creating visuals.

feel.exe resists the standard requirements of objectivity and effectiveness by making space for subjectivity, uncertainty and personal expression.

Built on a shader-based, stack-driven architecture, *feel.exe* accommodates both technical and intuitive engagement. Users with programming knowledge can shape more specific visual outcomes, while those unfamiliar with coding are invited to engage, led by their emotional expression, creating a tension between whether the code or the visuals are the main result of expression.

Figure 1. *feel.exe* example screen

The visuals respond non-deterministically: colours shift based on the amount of additional text written to explain a feeling; upper- and lowercase ratios influence animation speed. Missing information is filled in with values that randomly change over time. These non-deterministic elements

introduce instability and openness, foregrounding the tension between control and unpredictability, intention and interpretation.

Thus, it aims to raise critical questions about the translation of human feeling into code and proposes a more human, poetic perspective on programming languages.

feel.exe picks up on Lucas D. Introna's thoughts on what is seen as programmable, questioning if emotions can be used in programming and what is lost or added when they are used. It creates a new context surrounding code, resulting in a more intimate and personal experience while programming. Thus, it challenges the values surrounding code, showing how this changes what is created, how you create and who creates.

Conclusion

This research paper aimed to explore the interplay between programmatic code and society with a focus on the subjectivity of source code. The values embedded in the culture and history of Computer Science influence CS practices and therefore decide what its goals and responsibilities are. Even though code seems objective, it is thus largely influenced by its surroundings and social context. Research has shown how discriminatory values seep into algorithms, not just through the data but through the programming practices themselves, and thus have consequences when the resulting technology is applied to the real world. Through the strong mathematical influence on CS, we tend to extend mathematics rationality to algorithms, equating data directly with information, and ignoring what is lost and added in translation. The example of emotion recognition AI showed how history can influence the current applications of technology, visualizing how biases and problematic values persist even if the target group changes.

With source code forming the basis of technology, the context surrounding programming languages, the definition of programming languages and who is allowed or able to use programming languages, has a large influence on value encoding in technology. When goals and actions are translated into code, biases and values are embedded into the code, making it subjective even before it is executed. The medium we use changes what we create. Thus, the programming language changes what is programmable and the culture surrounding programming languages guides how a programming language should look like. This can lead to concepts like emotions having to be reduced and abstracted into something that is programmable, losing essential qualities in the translation process. To break open the value systems surrounding programming languages, feminist approaches like esoteric programming languages can be used that go against the normal conventions of how code should look. In particular multicoding languages challenge our assumptions of source code as the code and the output have two different meanings.

Exploring these topics through an artistic intervention in the form of a new esoteric programming language, where emotional expression and its subjectivity were harnessed as programmatic code, highlights the tension between code and its real-life influence. In *feel.exe* you codes visuals by writing about your feelings creating a tension between intention and interpretation as it questions what the main medium for expression is. Thus it breaks open the values and conventions surrounding code and programming languages and imagines a new coding practice.

Acknowledgements

This paper was written as part of the Journal Club hosted by César Escudero Andaluz and Sam Bunn at Interface Cultures at the University of Arts Linz.

References

Schlesinger, A. (2023). ADDRESSING COMPUTING'S DISCRIMINATION PROBLEM: A FRAMEWORK FOR ANTI-DISCRIMINATORY COMPUTING [Doctoral dissertation, Georgia Institute of Technology]

Bhattarai, P. (2017, October 19) Algorithmic Value: Cultural Encoding, Textuality, and the Myth of "Source Code". Catalyst, 3(1), 1-28. https://catalystjournal.org/index.php/catalyst/article/view/28786/21373

Chun, W. H. K. (2008, Fall). On "Sourcery," or Code as Fetish. Configurations, 16(3), 299-324.https://monoskop.org/images/c/cd/Chun_Wendy_Hui_Kyong_2008_On_Sourcery_or_Code_as_Fetish.pdf

Nagy, J. (2024). Autism and the making of emotion AI: Disability as resource for surveillance capitalism. New Media & Society, 26(7), 3989-4007.https://www.deepdyve.com/lp/sage/autism-and-the-making-of-emotion-ai-disability-as-resource-for-w55mQT7jBZ?#references

Shew, A. (2022, March). How To Get A Story Wrong: Technoableism, Simulation, and Cyborg Resistance. Including Disability, 1(1), 13 - 36. https://www.researchgate.net/publication/374400869_How_To_Get_A_Story_Wrong_Technoableism_Simulation_and_Cyborg_Resistance

Introna, L. D. (2011, November). The Enframing of Code: Agency, Originality and the Plagiarist. Theory Culture & Society, 28(6), 113-141. https://www.researchgate.net/publication/258192517_The_Enframing_of_Code_Agency_Originality_and_the_Plagiarist

Hermans, F., Schlesinger, A.(17 October 2024)A Case for Feminism in Programming Language Design, Onward! 2024: Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software(205-222). Association for Computing Machinery, https://dl.acm.org/doi/pdf/10.1145/3689492.3689809

Esoteric programming language. (2024). Esolang. Retrieved 05 07, 2025, from https://esolangs.org/wiki/Esoteric_programming_language

Temkin, D. (2020) The Aesthetics of Multicoding Esolangs. In Proceedings of the Electronic Literature Organization Conference 2020, 9. University of Central Florida. https://stars.library.ucf.edu/elo2020/asynchronous/proceedingspapers/17/.

Bibliography

Kneusel, R. T. (2022). Strange Code: Esoteric Languages That Make Programming Fun Again. No Starch Press.

Author Biography

Camilla Scholz is currently pursuing her master's at Interface Cultures at University Of Arts Linz, building on her background in HCI with a BSc in Creative Technology from the University of Twente. Working with creative coding and creating interactive installations, she imagines narratives of new perspectives. Currently, she is exploring esoteric programming languages and shader programming as artistic interventions. Her works have been exhibited at the Ars Electronica Festival (AT) and the Maker Festival in Enschede (NL).