Detangling Technology

A closer look at Computer Science education

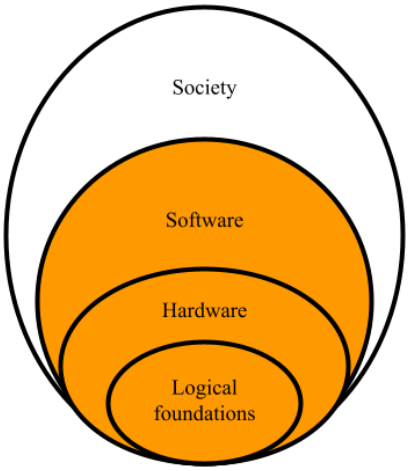
Clara Martens Avila, *BsC Computer Science @University of Amsterdam*, 27/10/2022

Technology[1] is an essential part of modern society: unless one chooses to live off the grid, some sort of interaction with it will have to take place. But what kind of interaction occurs, and what level of understanding is necessary between humans and our technology, is still up for debate.

In this essay I will examine the state of the Computer Science education ecosystem. I would like to start by proposing five categories of people that require knowledge about technology:

1. The general public, who need to be aware of how to use computers and technology safely[2]
2. The academic world outside of Computer Science, who want to discuss the influence of technology on their own field (be that sociology or business administration)
3. Professionals who need knowledge of technology to do their job correctly, be it for policy making, management, exploitation or discussion
4. Aspiring programmers, who want to learn how to code to build products
5. Aspiring Computer Scientists, who beyond just coding want to understand the building blocks of code, computers and its applications

These wants and needs are provided for in a complicated network of bootcamps, traineeships, online courses, academic and non-academic institutions, elective courses and re-training programs. But the bottom line in most offerings seems to be: if you want to join the conversation, learn how to code. This certainly benefits companies desperate for programmers. But does one need to know how to paint to go to a museum or appreciate art? What if one doesn't have a talent for painting (or programming)? Does one need to know the process used to make a knife, to learn how to use it carefully and to know that the edge is sharp? How does programming knowledge tie into the discussion around applied sciences versus the academic world? And what exactly is technology?



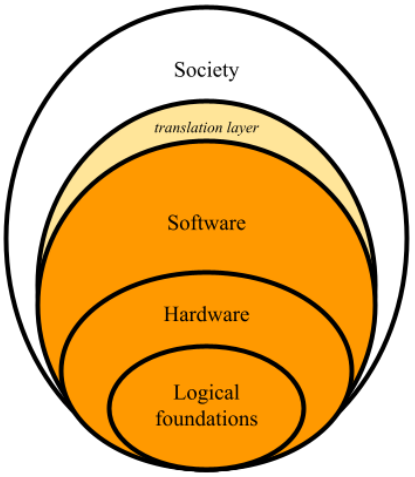
Let us start with that last question. It serves as a bridge to the other questions. A computer scientist would start explaining their field by defining hardware and software: the distinction between the inner workings of the computer and the art of writing programs that run on it respectively. This, in conjunction with the theoretical mathematics at the base of it, are indeed the layers of the "russian doll" egg (see image) that a computer scientist would concern themselves with. A programmer would be operating more on the software layer, a computer architect on the hardware layer.

Essential to this depiction here is that if someone operating on one layer ever wonders why things have to work the way they do, the answer will usually lie in the layer under them. This is true all the way down. The most difficult step however is in the translation between the encapsulating layer, "society", and the layers under it. A manager learns how to manage and help grow an IT team, a sociologist learns how to write a paper on filter bubbles: both approach the field from their own discipline. And as much as computer scientists would like to deny it, *this societal layer is essential to the concept of technology.* A big part of what is colloquially known as "the tech industry" is not even concerned with programming or computer architecture: the startup founders, marketeers, crypto grifters and project managers are as much a part of the tech ecosystem as the developers are, if not more. Programmers are the equivalent of the builders of a house designed by an architect that may or may not know less about designing houses than they do.[3]

It seems common sense to make all other stakeholders in the process of building this house more conscientious. But does teaching the architect and inhabitants of the house the basics of laying one brick on top of another help the system? Probably not. So how do we make sure the building doesn't crumble underneath us?

I think the solution is threefold. First, we have to put a lot of emphasis on the distinction between technology and the societal meaning of technology: the inner and outer egg. We have to acknowledge the translation layer between the "objective" tech and "subjective" use of it[4]. No more broad use of the word "tech"! Let tech be the inner egg, and let the humans and organisations around it be the ones that frame the interpretations and applications of (more) neutral technological concepts[5]. This levels the playing field for discussions between the outer and inner layers. It is important that this levelling goes both ways: computer scientists should not be able to hide behind technical definitions and concepts to excuse the mismanagement and misuse of the products they come up with. Likewise, the outer layer should be very mindful of the perspectives they approach technology with.

Secondly, we have to be open to new ways of teaching the "inner egg". Being mindful of the translation that takes place, we must still try to make that translation layer as thin as possible. As discussed, the main way to bridge this gap employed right now is to teach non-computer scientists how to code. Almost all Business (Information) Administration courses in The Netherlands offer some sort of "introduction to programming" course. I would argue that as a programmer, the only thing worse than a manager that knows nothing about software development and infrastructure, is a manager who knows how to declare a function and is at the very worst point of confidence-ability ratio in the Dunning-Kruger effect (Dunning, 2011). A parallel can be drawn here with the productivity paradox: more investment in (in this case) tech education does not necessarily result in better IT managers (Brynjoffson, 1993). A common hypothesis to explain the productivity paradox is IT mismanagement. But programmers do want managers who can support them properly! How do we teach these managers about the inner egg without coding?



It is important to remember that the inner egg is more than the sum of its parts: Computer Science, as a scientific field, is a way of looking at the world, of solving problems and approaching challenges. It is a mindset that makes it very easy for a computer scientist to pick up new programming languages, to understand the thousands of new libraries that come out every year for their field of choice, and to have such an abstract understanding of technological concepts that they can apply it to any situation. Why focus on teaching the outer egg software development when we can teach people *to think like a computer scientist*? Computer Science is full of fundamental building blocks that one does not need to have worked with to understand!

Take for example an API[6]: a concept encountered by many, understood by few, very useful in discussions around data, privacy and new stories like the Cambridge Analytica Scandal. Do people need all the different physics models and formulas used to define gravity to understand the concept? No. I propose teaching about Computer Science through building blocks like APIs in a conceptual but applied way, instead of programming.

Third: in an academic sense too we have to be very clear about the distinction between the practical field of software development and the scientific field of Computer Science. There has been a push in recent years to make university students more "employable" by teaching them practical skills. Computer Science, already a very practical program, seems like an ideal candidate.

The issue here seems to be the societal stigma around practical education. Software companies, hiring in a very practical field that can easily be self-taught, still give a lot of weight to university degrees (Mazzina & Dene, 2016). Parents want their children to go to university (Heimlich, 2012)! This is particularly remarkable considering first-world countries like The Netherlands are experiencing a huge demand for applied sciences and practical skills (and people who do possess these skills are rewarded accordingly).

We should give more importance to the theoretical value of Computer Science and export that to other courses, academic disciplines and societal movements wanting to include technology in their curriculum. That is how we create an even playing field in which all layers can communicate and solve problems together, each with at least an understanding of the reasoning of the others. Simultaneously, we need to make clear that software development is a valuable career path, through whatever bootcamp or university study one chooses to approach it.

At the beginning of this article I identified five categories of people looking for technological knowledge. I hope to have made clear that "learning how to code" is not the one-size-fits-all solution that it seems, and that technological awareness can be presented in any number of ways. The general public does not need to (and mostly does not want to) know how to code a website together, but they would benefit from knowing what trackers are beyond "Facebook listening to everything you do". Not every developer would benefit from a university education, and not every computer scientist needs to be a developer. Those who discuss and work with technology would benefit more from some conceptual Computer Science over learning a few programming basics and approaching tech problems from their own discipline otherwise. In conclusion: technological education (to those not seeking to become computer scientists or programmers) can be simultaneously less in the software layer, and less in the societal layer.

Notes

- The “egg” graphs are inspired by “the egg” describing science in *Scientific Methods in Computer Science* (2022). They are of the author’s own creation though.
- The author of this piece is finishing up her Computer Science degree, has more than six years of experience in the tech industry and has both participated in, ran, and taught 6-week part-time web development bootcamps through Turing Society. She has also followed various courses from social and information sciences.
- This article has mostly discussed “the egg” in a top-down manner. The author has hinted towards making computer scientists more aware of the upper layer as well, but this bottom-up approach will have to be expanded on in a future article.

1. Technology, in this article, is used as “digital technology”: everything from computers to smartphones and all the networks and programs those entail.
2. The word *need* is used here, because not all people are necessarily very aware of or interested in the dangers of technology.
3. We could go further with this analogy and bring “software-development only” projects into the mix like a lot of open source software: great project concepts, excellent software execution, but little to no attention to other aspects necessary for the success of a product like marketing and UX. This would be the equivalent of a very functional but ugly, inhospitable building.
4. This is, of course, a far more complicated topic, for what is truth and what is objective? See also: discussions on post-modernism and truth seeking like *Post-Truth* (2018).
5. I truly believe this, from a philosophical perspective. Even the technological inventions that I consider most heinous, like NFTs, I still consider neutral (though useless) in their technical definition and horrendous in their applications.
6. Short for application programming interface, used to request data and processes from other programs without accessing the source code. An API can be used for example to access map data from a geographical organisation, without accessing their database directly and ensuring the organisation has control over the API usage.

References

Brynjoffson, E. (1993, December). The productivity paradox of information technology. *Communications of the ACN*, *36*.

Dodig-Crnkovic, G. (2002). Scientific Methods in Computer Science.

https://www.researchgate.net/publication/2563629_Scientific_Methods_in_Computer_Science

Dunning, D. (2011). The Dunning–Kruger Effect: On Being Ignorant of One's Own Ignorance. *Advances in Experimental Social Psychology*, *44*. https://www.sciencedirect.com/science/article/abs/pii/B9780123855220000056?via%3Dihub

Heimlich, R. (2012, February 27). *Most Parents Expect Their Children to Attend College*. Pew Research Center. Retrieved October 27, 2022, from

https://www.pewresearch.org/fact-tank/2012/02/27/most-parents-expect-their-children-to-attend-college/

Mazzina, A., & Dene, K. (2016, October 7). *Do Developers Need College Degrees?* Stack Overflow Blog. Retrieved October 27, 2022, from https://stackoverflow.blog/2016/10/07/do-developers-need-college-degrees/

McIntyre, L. (2018). *Post-Truth*. The MIT Press.