Software Development vs. Computer Science

2009

2 min read • 429 words

Themes: Programming Human Centered Spiritual

Most developer job applications that I see have a "BS in Computer Science or equivalent experience" requirement.

During my studies in Computer Science at George Mason University, though short, I learned a number of things. One of them was what a waste it was to learn such higher math in my field. I want to develop software, not develop the most cunning-edge earth-shattering algorithms

> This distinction between software development and algorithm design would later influence Kenneth's "for humans" philosophy—focusing on practical, usable software rather than academically impressive but complex solutions.

. I don't want to reinvent ssh or find a better way to implement pgp keys. Of course, those things are necessary in certain fields, but not in mine.

I was taught that Programming is to Computer Science as a Telescope is to Astronomy

> This analogy reveals a fundamental misalignment in computer science education. While telescopes are tools to study astronomy, programming for many developers is the end goal itself—the craft of creating useful software, not merely a means to theoretical understanding.

. Its a tool to get to a means. If this is true, than why wasn't I taught how to make software?

Many Software Developers also have degrees in Electrical Engineering. Why is this? A CS degree doesn't seem to quite fit either. Perhaps we should make a separate degree for Software Development?

Computer Science should be separated from Software Development. They should be two different Degrees

> Kenneth's prescient call for separating CS from software development anticipated the emergence of bootcamps and practical coding programs. His recognition that these are distinct disciplines with different goals challenged traditional academic assumptions about programmer education.

.

When I realized this, I started spending my time focusing on design rather than math. I learned a great deal of things from color theory to relational spacing and I found myself a new home: web design. It's a beautiful field. I started to spend my own time learning software development, rather than spending hours studying Calc 2

> Kenneth's shift from theoretical math to practical software development and design fundamentals shaped his later success. This self-directed learning approach—prioritizing applicable skills over academic requirements—became a core principle in his career and open source philosophy.

.

Fifteen years later, this intuition about the mismatch between computer science education and practical software development proved prescient. The same focus on human-centered design over academic abstraction that led me to abandon calculus for color theory would eventually inform the 'for humans' philosophy and my understanding of programming as spiritual practice—recognizing that the most profound technical work serves human flourishing rather than algorithmic elegance.
