# What is Computer Science? Efficiently Implementing Automated Abstractions

February 2010 (Ph.D. student)

**Summary**

This article elaborates on my four-word summary of the essence of Computer Science as a field of study: *Efficiently Implementing Automated Abstractions*. I also provide one-, two-, and three-word summaries for those who want to be more concise.

## INTRODUCTION

Many people who study Computer Science (myself included) have trouble explaining what exactly it is to people who are not in the field. This article is my attempt at defining what *Computer Science* is as a field of study, in four words or less.

Apologies in advance if this article sounds overly fluffy or bullshitty; I don't often get to pontificate about such lofty abstract ideas. Adults who have real jobs probably think that since I'm working towards my Ph.D. in Computer Science, I must spend all day sipping lattes and engaging in philosophical discussions with my colleagues in some Ivory Tower; in reality, I spend most of my time 'in the trenches' hacking on research prototypes and debugging memory errors in C code. So here comes my rare attempt at being 'scholarly' ... enjoy!

## ABSTRACTION (ONE-WORD SUMMARY)

It's nearly impossible to summarize an entire field in one word, but if I had to choose just one for Computer Science, it would be ***abstraction***. At its core, Computer Science is about building clean abstract models (abstractions) of messy, noisy, real-world

Philip Guo

Assistant Professor of Cognitive Science

UC San Diego

Publications
Google Scholar
Curriculum Vitae

Twitter: @pgbovine
philip@pgbovine.net
read before cold-emailing

Want to join my lab?

Office Hours

UCSD Cognitive Science
9500 Gilman Drive
La Jolla, CA 92093-0515

Newest RSS

PG Vlog #11 - Playing Opens the Door to Serendipity

PG Vlog #10 - Managing is Nudging

PG Vlog #9 - For Aspiring Creators, Content Means Everything

PG Vlog: why I'm trying out videos

PG Vlog #8 - Starting Creative Projects in Private

PG Podcast - Episode 20 - Cat Hicks on ethical data science

PG Podcast - Episode 19 - Jean Yang on time and serendipity management

objects or phenomena. As Computer Scientists, we must choose what to include in our models and what to discard, to determine what is the minimum amount necessary to model in order to ***solve our given problem*** to the required degree of accuracy.

Computer Science is a means of solving real-world problems, and it all starts with abstraction. For example, the first step in building an automated movie recommendation system (like what Netflix does) is to choose what features of movie watcher behavior to model, to *abstract* a movie watcher to a set of relevant metrics (e.g., number of times he's rented Arnold Schwarzenegger films). Ideally, we would like to fully model the human brain so that we can make near-perfect recommendations, but that's obviously intractable (for now, at least).

## AUTOMATING ABSTRACTIONS (TWO-WORD SUMMARY)

The word I would immediately add to my summary is ***automating***. Related fields like mathematics and the natural sciences also involve building abstractions (e.g., of geometric shapes or atomic structures), but their models only serve to describe and explain, not to evoke actions. What makes Computer Science different is that it deals with putting the models into action to solve problems. This involves creating *algorithms*, which are step-by-step instructions for performing actions on and with the data that we have modeled.

In our movie recommendation example, once we have formed the proper abstractions (models), then we need to figure out how to act on them in order to make recommendations. One very simple algorithm would be to simply recommend more Arnold movies to people who have rented Arnold movies in the past, or perhaps to throw some Stallone movies into the mix.

I've purposely not chosen *automating* as my first word, since automation existed far before the invention of Computer Science. People have thought about automation for thousands of years, starting with building tools to automate aspects of farming and culminating with the assembly lines of the Industrial Revolution. Instructions for assembling cotton gins or guns or automobiles are definitely *algorithms* for automating tasks, but they deal with the world of concrete, real-world objects; no abstraction is needed.

PG Podcast - Episode 18 - Michael Kennedy on podcasting as a career

A Five-Minute Guide to Ph.D. Program Applications

PG Podcast - Episode 17 - Tess Rinearson on dropping out of college and into the tech industry

[see all articles]

Categories

research (40)

jobs (40)

assistant professor life (38)

programming (37)

software (35)

Ph.D. student life (32)

productivity (31)

computing education (26)

personal (26)

social observations (26)

PG Podcast (22)

undergrad education (20)

human-computer interaction (14)

talks (14)

CACM blog (13)

On the Move memoir (12)

kids (12)

PG Vlog (12)

guest article (11)

photography (11)

Asian parents (11)

research advising (10)

faculty job applications (9)

teaching (9)

writing (8)

email (8)

screencast video (7)

high school (7)

In contrast, abstraction is *essential* for solving problems like effectively finding information online, predicting stock prices, optimizing flight plans to save gasoline, or automatically detecting credit card fraud; that's part of the reason why these are Computer Science problems (rather than, say, industrial or process engineering problems).

## IMPLEMENTING AUTOMATED ABSTRACTIONS (THREE-WORD SUMMARY)

Notice that I haven't mentioned computers at all so far; that's because, at its core, Computer Science doesn't involve computers at all. You could imagine creating the requisite models and algorithms, then handing them off to a team of people to execute in an assembly line filled with pencil and paper. That would still be doing Computer Science per se, but it would be too slow to solve any practical problems.

Computers are electronic devices that are engineered to be amazingly fast at executing algorithms on data, so to solve any real Computer Science problems, we need to ***implement*** our models and algorithms (the 'automated abstractions') in the form of code (instructions) that the computer can understand.

This is where the rubber hits the road. So far we've been dealing in some intangible fairy world of abstract models and algorithms. We might jot our notes down on paper, formalize them and do mathematical proofs about their properties, or write scholarly papers trying to persuade others as to why our chosen models and algorithms ought to work well. However, the best proof that our proposed solution truly works comes from actually ***implementing*** it in the form of a computer program, executing it on a computer, and using the output to *affect the real world*. No matter whether the output is shown to a person (e.g., Google search results) or fed into a mechanical device (e.g., airplane autopilot system), it has a direct, tangible effect on the world. If an implementation actually works, then nobody can argue that it doesn't work (this statement sounds silly, but you really can't achieve that high degree of certainty that you're undeniably correct in most other life endeavours).

The beauty of having a properly-functioning (correct) implementation is that it ***cuts away all the subjective bullshit***. As a user, the only way that Netflix can convince you that its recommendation system works is if it actually provides

good recommendations for you; without a working implementation, no amount of persuasive rhetoric from the CEO or even mathematical proofs from resident theorists (which might contain logical flaws or unrealistic assumptions) can convince you otherwise.

## EFFICIENTLY IMPLEMENTING AUTOMATED ABSTRACTIONS (FOUR-WORD SUMMARY)

The final word I'm piling onto the summary is ***efficiently***. First and foremost, we must ensure *correctness* in our implementation; it doesn't matter how fast your code runs if it doesn't properly solve the problem. Next, we can think about *efficiency*, designing our data models and algorithms to run quickly while taking up the minimal amount of resources (e.g., memory, hard disk space, electricity). Computer Science researchers have developed many theoretical and empirical techniques to make implementations of algorithms more efficient.

So how efficient is 'efficient enough'? Well, it depends on your particular application. If your movie recommendation algorithm must run for 1 year before giving results to the user, then it's useless. (Actually, if you ran today's Netflix or Google algorithms on computers built 20 years ago, they would probably actually take a year to run!) But the difference between it taking 0.5 seconds to run and 0.005 seconds (a 100X factor) might not matter to most users.

## RECAP

Again, here are the 4 words I would use when describing the essence of Computer Science, in order of significance:

1. Abstraction
2. Automating Abstractions
3. Implementing Automated Abstractions
4. Efficiently Implementing Automated Abstractions

Note that I purposely didn't use either the word 'computer' or 'science' in my summaries, since I don't think they properly embody the essence of the field. I'll finish with my thoughts on those two words, though:

***Computers*** are merely a tool for *implementing* Computer Science ideas. Edsger Dijkstra, a pioneer of the field, once said, *"Computer Science is no more about computers than astronomy is about telescopes."* That said, though, since computers are the link between Computer Science and the real world, lots of Computer Science research is actually geared towards improving the capabilities of computers! Researchers and engineers write computer programs to help them build more powerful next-generation computers (with faster processors and more storage); this creates a wonderful positive feedback loop where the next generation has more powerful computers and so are able to implement even more ambitious Computer Science ideas, including ideas to improve future generations of computers. Also, lots of Computer Science research is geared towards helping people interact more productively with computers; if computers didn't exist, then there would be no research in sub-fields like programming languages, operating systems, or human-computer interaction.

And as for ***science***, I believe that one must constantly use the scientific method of hypothesis creation, testing, and refinement when designing, implementing, debugging, and tweaking implementations of Computer Science ideas. Nobody designs or implements an idea correctly on the first attempt, so that's why a methodical, empirically-driven scientific mindset is required to create correct and efficient implementations of Computer Science ideas.


## ACKNOWLEDGMENTS

Many of the ideas in this article were inspired by the final lecture of the MIT 6.00 class given by Professor John Guttag, entitled What do computer scientists do?.



Created: 2010-02-04
Last modified: 2010-03-06



Related pages tagged as *computing education*:

- PG Podcast - Episode 13 - Lindsey Kuper on a new kind of computing conference
- PG Podcast - Episode 11 - Brad Miller on building long-lasting software in academia

- Python Tutor: The First Three Years
- Interactive Systems for Learning Programming at Scale
- Learning programming at scale
- My CS Education Zoo interview
- My Basic Technology Stack for Teaching Web Programming
- Programmers: Please don't ever say this to beginners ...
- Hello World in C and Python
- Python is now the most popular introductory teaching language at top U.S. universities
- Python Tutor Live
- NPR Interview on Silent Technical Privilege
- My book notes on Unlocking the Clubhouse
- Silent Technical Privilege
- Hour of Code: Observations from a Middle School Classroom
- My Unexpectedly Awesome AP Computer Science Class
- Hacker School Residency
- Teaching Librarians Programming
- Education Removes Fear: Some Examples From CS Courses
- Teaching Real-World Programming
- Teaching Programming To A Highly Motivated Beginner
- Introductory Computer Programming Education
- Why Python is a great language for teaching beginners in introductory programming classes
- Computer Science in Modern Everyday Life
- Java and Software Engineering Notes