(or the art of unassuming titles)

Henri Casanova
ICS Graduate Chair
(henric@hawaii.edu)

Motivation

- I’ve been hosting ICS690 for 8 semesters, and I still don’t know what CS is!
- I don’t think I even have my own definition of what it is
  - I could come up with one, perhaps
  - But likely not a good one anyway
  - And very biased by my research area
  - In fact, each professor in this department and others would likely give a different definition
  - I figure smart people have surely come up with a good one, so why bother... I can Google it if needed one day

Motivation

- Not knowing what one's own field is sort of a problem....
- Wikipedia: "the scientific and mathematical approach to computation and to the design of computing machines and processes"
  - Sounds good enough, has a science, mathematics, and engineering vibe
  - How did this kind of definition come about?
- Turns out, many hard questions need to be answered to understand what CS is
  - Beyond just giving a definition

Some Questions?

- “What is the subject matter of CS?”
  - Algorithms, computers, programs, people, everything?
- “What kind of methods do computer scientists use to do their investigations?”
  - Hot topic among the faculty: “we should teach a methods course!” “I know nothing about your methods and I don’t care about them!”
- “It is a science?”
  - “Anything which has to call itself a science, isn't.” [Brooks, 1996]
  - Is the term just added for respectability?

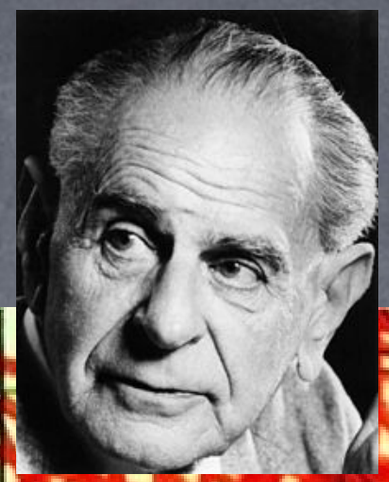
Reading Group

- The last two semesters, we started a small reading group (Mike, Lisa) with the goal of understanding “what is science?”
  - That’s right, not ambitious either...
- We read books about the philosophy of science
  - “What is scientific knowledge?” (epistemology)
  - “What are scientific methods?”
  - “What are the limits of scientific knowledge?”
  - “How does science develop?”
  - etc.

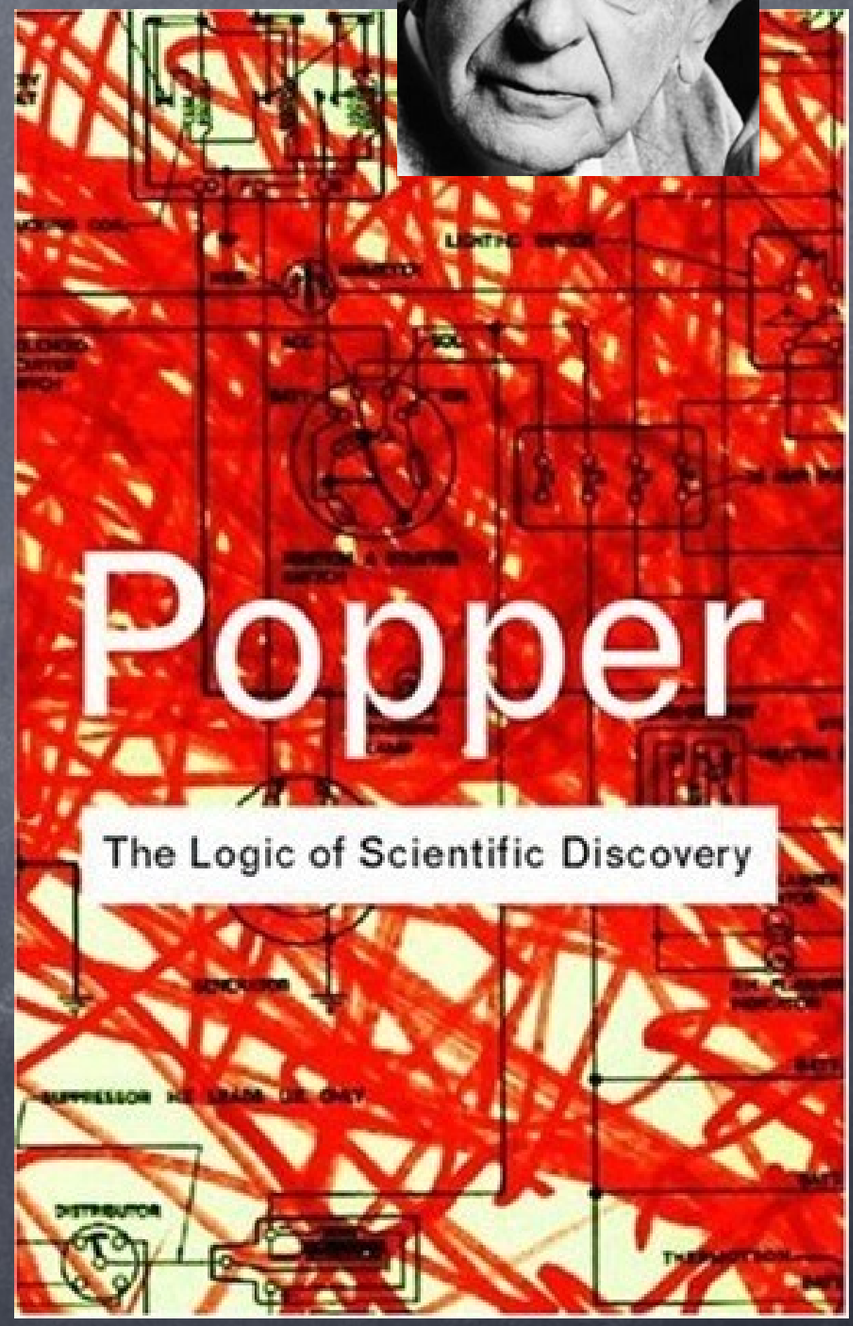
Philosophy of Science

- That branch of philosophy started early 1900s, starting with positivism
- Positivism: knowledge comes from experience, and scientific claims are meaningful only if verified in experience
- Sounds good, but: How do we verify?
  - e.g., “All matter consists of atoms”: are we going to check the universe?
  - We need some leap of faith to declare “ok, fine, this is a law that applies always”

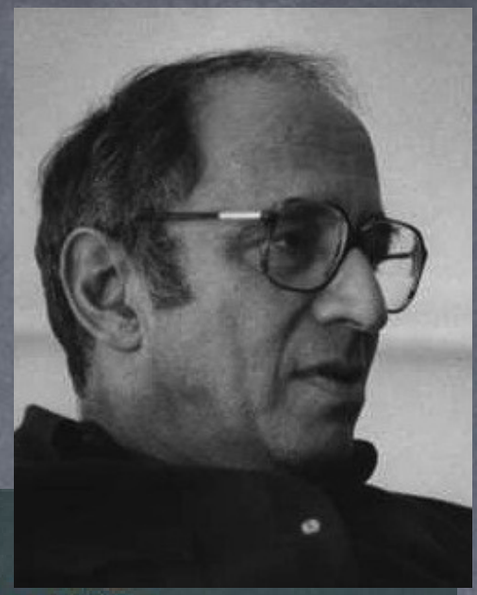
Karl Popper



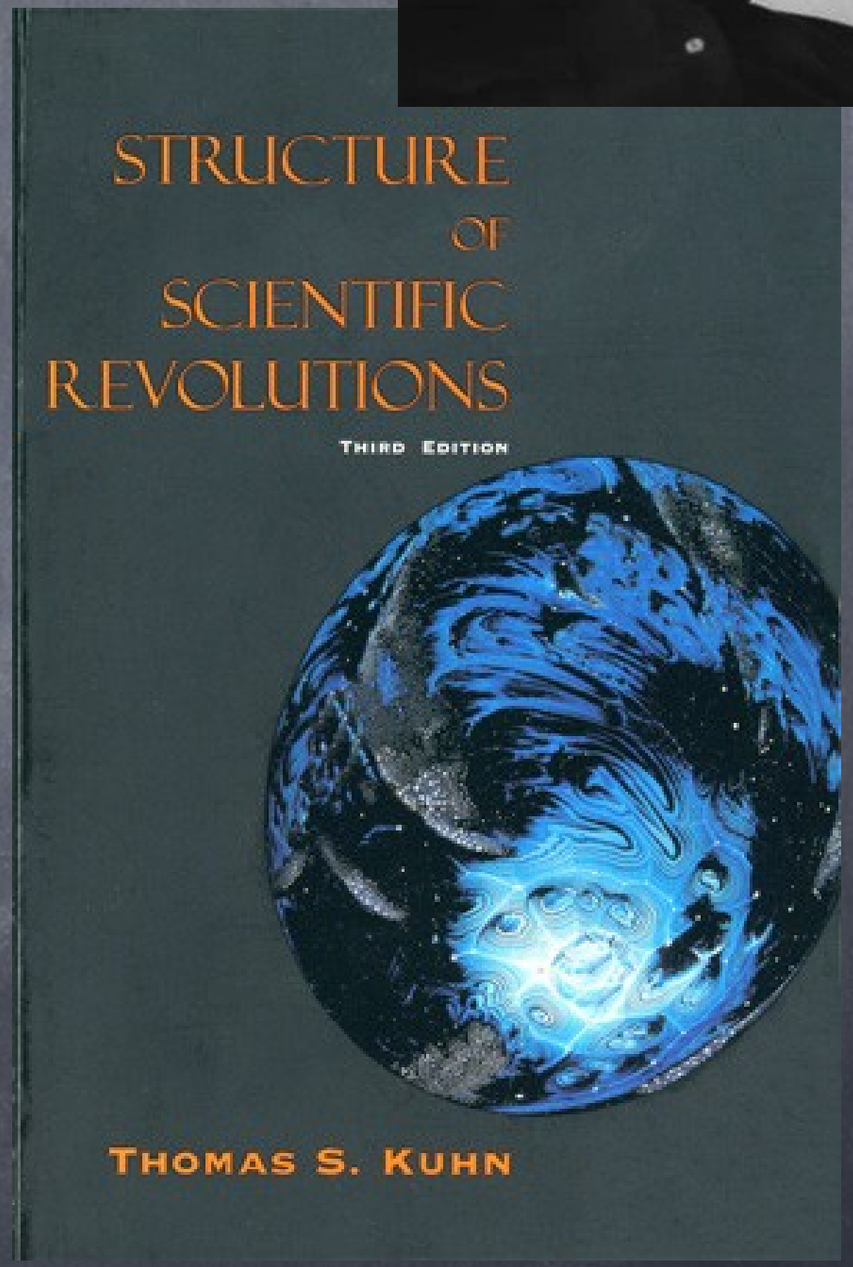
- Scientific statements should not be verified (because they can’t be) but instead falsifiable!
- Statements have simply to “prove their mettle” until proven wrong, at which point better statements are sought
- No absolute scientific truth, only scientific truth for now...



Thomas Kuhn



- Science proceeds as: “normal science” - crisis - revolution - “normal science” - crisis - revolution - etc..
- Normal science: all is well in our theory of the world
- Crisis: some experiment proves that our theory doesn’t always work!
- Revolution: intense period in which new theories are proposed



So what?

- We learned several things, and perhaps changed our way of thinking about these matters
  - I wrote my first “popperian” paper
- As expected, those books don’t solve everything and in fact it’s fair to say that there is some disappointment
- So now I understand a tiny bit better what science is... but I still don’t know what CS is
  - Some parts clearly are not science, and some parts may be

Is CS Science?

- “What is so special about scientific knowledge is that it is derived from facts, rather than being based on personal opinion” (Chalmers, 1999)
- What are facts in computer science?
  - Computers are man-made, so are there timeless laws as in physics?
  - The things that physicists study happen without scientists
  - Do the subject matters of CS exist without computer scientists? I guess it depends....
- What is new knowledge?
  - If in biology you observe something new about some organism, you’ve added to the body of knowledge
  - If in CS, you observe something new about some system that some grad student built, do we care? It depends....
  - Hence the difficulty of defining new research projects!

Is CS Mathematics?

- Mathematics is not an (empirical) science
- It’s about man-made objects/concepts
- So is CS
- Many would agree that CS is mathematics
  - Turing, Dijkstra, knuth, etc.
- There is a strong mathematical tradition in CS
- But nowadays, it’s clear that mathematics is no longer fundamental for many CS activities
  - We’ll see how that has come about....

Is CS Engineering?

- Engineering: about making things, not about explaining the world
  - “The scientist builds in order to study, the engineer studies in order to build” [Brooks, 1996]
- Many CS researchers build systems and test them, just like engineers. These systems comprise man-made computers.
- Strong opposition from the mathematical tradition: “Computer science is no more about computers than astronomy is about telescopes” [Dijkstra, but in fact Fellows]
  - computer science ≠ computer engineering

So, which is it?

- Clearly, CS is sort of like math, sort of like science, sort of like engineering
- This is why it seems so all-over-the-place to outsiders
  - Look at our faculty
  - Nobody on campus can understand what we do
  - It would be great to be able to tell them!
- Is CS a loosely connected set of computing-centered fields and just an umbrella (and thus not useful) term to even have?
- Or should we have a (good) definition?

Definitions

- Too broad: “the study of phenomena surrounding computers” [Newell, Perlis, Simon, 1967]
  - It describes what Computer Scientists do, but if computers are a product of CS, then it’s a circular definition
    - Like “Mathematics is what mathematicians do”
- Too narrow: “Computer Science is the study of the theory and practice of programming” [Khalil, Levy, 1978]
  - Is CS merely programming????
- We’ll look at other definitions

Should we care? (I)

- Hamming believes it's important not to ignore this question ("What is CS?") and that it's a bad idea to just "get on with doing whatever it is we're doing"
- "The picture which people have of a subject can significantly affect its subsequent development. Therefore, although we cannot hope to settle the question definitively, we need frequently to examine and to air our views on what our subject is and should become" [Hamming, 1969]

Should we care? (II)

- Studying what CS is highlights the diversity of the field and the challenges that diversity brings with it
- Heated arguments about how computer scientists should work have roots in different conceptions about what the field is
- Many misunderstandings/controversies can be avoided by knowing the philosophical traditions
  - “How can you be a computer scientist and not know this????”
- Very difficult to come up with common understanding of how research should be done
  - e.g., my interactions with Prof. Suthers or Prof. Poisson
- Are CS department just wildly different people who happen to be on the same floor?
  - A small department showcases this very well :)

Some Philosophical Questions

- “Do computer scientists prove formulas like mathematicians, build things like engineers, test hypotheses like scientists, do all the above, or do something else entirely?”
- “What is progress in CS?”
  - Look at list of Turing award recipients
- “Are new algorithms or programs progress?”
- “How does new knowledge about computing becomes a part of commonly held knowledge about computing?”
- “What kinds of things in CS are universal, objective, timeless?”
- “Are algorithms abstract or concrete? Is everything a computer?”
- etc.

- Great to discuss over a beer, or in this seminar, but what would people who study the philosophy of (computer) science say?

Philosophy of CS

- The Philosophy of CS is not well developed
- I have no idea what it is
- So I figured: great topic for the seminar
- First thing to do: what have people done about this?
- Answer: Matti Trede teaches a full semester course on the philosophy of CS in Finland
- So, as a first step (toward such a course? an ICS690 “module”?) I’ve read his lecture notes and material and will shamelessly use this as a basis for my rambling
  - Already have in the previous slides
- You are thus guinea pigs of this horrible experiment
  - Feedback, participation MOST welcome here

Outline

- History of the identity of the field
  - Funny / illuminating quotes abound
- Discussion of the three CS traditions
  - math, science, engineering
- Fundamental question

History of the Discipline

- CS has a short history
- The question “what’s the first computer” has no clear answer
  - Leibniz, Pascal, Babbage, Ada Lovelace, Jacquard Loom, Zuse, ...
- But what is the first “stable” part of CS as a field?
- Likely: the stored program paradigm, the set of innovations around that paradigm
  - The Church-Turing thesis (without this, some argue that there would be no field of computing, just eclectic knowledge about particular machines)
    - Turing Machine
  - Von-Neumann architecture
    - ENIAC: the first Turing complete machine (1946)
- So, right after WWII
- In 1958, in Communications of the ACM: “What is your reply when someone asks your profession? Computing Engineer? Numerical Analyst? Data Processing Specialist?”
  - asked for input on “What should we name this field?”

1940s - 1970s

- In 1946, first society for computing professionals: “Subcommitte on Large-SCale Computing” (part of AIEE (Am. Inst. Electrical Eng.))
- In 1947: Foundation of the ACM
- In 1951, the IRE (Inst. Radio Eng.) formed “Professional Group on Electronic Computers”
- In 1963: AIEE and IRE merged, becoming IEEE and creating the “IEEE Computer Society”
- IEEE: Standards and hardware
- ACM: Theoretical computer science and applications
- Or at least that was the idea
- It’s not clear if in those years members of ACM/IEEE thought of themselves as “Computer Scientists”

1940s - 1970s

- In the 1950s, Communications of the ACM suggested names for people in the field
  - turingineer, turologist, flow-charts-man, applied meta-mathematician, applied epistomologist, comptologist, hypologist, computologist (this last one had quite a bit of support)
- By the turn of 1960, hot debate about the name of the field
  - likely meaning that it had really become a field
- By the late 1960s, the name “Computer Science” had been accepted
  - 1962: first CS department (Purdue Univ.)
- Knuth, 1985: “I suppose the name of our discipline isn't of vital importance, since we will go on doing what we are doing no matter what it is called; after all, other disciplines like Mathematics and Chemistry are no longer related very strongly to the etymology of their names.”
- Dijkstra didn’t like it: “It’s like calling surgery Knife Science”
- But it still wasn’t clear to people what the field was, even though it had been named
- In 1967, we have a first “official” definition...

One Definition

- Allen Newell, Alan Perlis, Herbert Simon, Science, 1967: “Wherever there are phenomena, there can be a science to describe and explain those phenomena. Thus, the simplest (and correct) answer to ‘What is botany?’ is ‘Botany is the study of plants’. And zoology is the study of animals, astronomy the study of stars, and so on. Phenomena breed sciences. There are computers. Ergo, computer science is the study of computers. The phenomena surrounding computers are varied, complex, rich. [...] Computer science is the study of the phenomena surrounding computers.”
- This definition leads toward empiricism
  - observe, theorize, invalidate, repeat
  - implement two algorithms, compare them, draw conclusions
  - send packets through the internet, compute node degree, observe power laws, draw conclusions

Is it useful?

- The authors of this definition argued that the study of computer “does not lead to user sciences” but instead “to the further study of computers”
  - The computer is not “just an instrument” but a “phenomenon”
- Begs the question: is this field useful to outsiders?
- Many objections for this definition
  - “the phenomena surrounding computers” is way too broad
  - is it a good idea to focus on the computer?
    - What “computer” means will change
    - And what about algorithms, programs, users
  - And aren’t computers only instruments?
    - See Dijkstra’s point of view

Forsythe's Definition

- George Forsythe: "I consider computer science, in general, to be the art and science of representing and processing information and, in particular, processing information with the logical engines called automatic digital computers."
- This definition focuses on computing, not computers
- Note the terms "art" (meaning "craft") and "science"
- Knuth: "Science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it's an art to deal with it"
- Is CS a craft or a science?
  - Is programming a craft or a science?
  - If it's a craft, then is it even part of CS? (if CS is supposed to be a science)
- But Forsythe's definition tries to cater both to the pragmatic and abstract side of CS

Theory vs. Practice

- At the turn on the 1970s there were two camps
  - theoretical CS: computational complexity, logic, etc.
  - practical CS: computer architecture, programming, etc.
- In 1962, the ACM started making curricular recommendations and the first draft was 1965
- In 1967, the ACM said: “The subject areas of computer science are grouped into three major divisions: information structures and processes, information processing systems, and methodologies”.
- In 1968, the curricular guidelines include “...these recommendations are not directed to the training of computer operators, coders, and other service personnels.”
- Follows strictly the mathematical tradition, and therefore:
  - Programming itself is not part of CS
  - Programmers are not computer scientists (but computer scientists often program)
    - In fact, I get this question a lot from “laypersons”!
- In 1974 the NSF recognizes CS as its own discipline

Informatics????

- In 1970, the International Federation for Information Processing (IFIP) coined Informatics: “the science of the systematic and effective treatment (especially by automatic machines) of information seen as a medium for human knowledge and for communication in the technical, economic, and social fields.
- CACM, 1998: “In fact there is no clear agreement even on the name of the field. In European universities, the titles of many of the relevant departments revolve around the word ‘informatics’, whereas in the US most departments are ‘computer sciences’. To avoid using the name of the machine, some use the word ‘computing’ instead”
- Dijkstra preferred “computing science”...

Quid of Programming?

- 1972, CACM, Dijkstra wrote: “We must not forget that it is not our business to make programs; it is our business to design classes of computation that will display a desired behavior”
- Instead of proving program correctness once it is written, Dijkstra felt that one should build correct programs from the ground up (“mathematical proof by construction”)
  - This point of view never gained much support outside of academia
- But there is nevertheless a strong mathematical tradition in CS

Go Mathematics!

- In the 1970s, many Computer Scientists felt that to legitimize the field , it would have to be recognized in the mathematical community
- Donald Knuth, 1974, American Mathematical Monthly: “Like mathematics, computer science will be somewhat different from the other sciences, in that it deals with man-made [sic] laws which can be proved, instead of natural laws which are never known with certainty. [...] The difference [between mathematics and computer science] is in the subject matter and approach — mathematics dealing with more or less with theorems, infinite processes, static relationships and computer science dealing more or less with algorithms, finitary constructions, dynamic relationships.”
- Dijktra wrote a similar article but talked about how what’s taught in mathematics is actually detrimental to the development of good computer scientists

Go Mathematics?

- Dijkstra's view on Math vs. CS
  - Standard collection of concepts vs. concept-creating skills
  - Learning standard notations vs. inventing ad-hoc notations
  - "In the standard mathematical curriculum, the student often only sees problems so 'small' that they are dealt with at a single semantic level. As a result many students see mathematics rather as the art of organizing symbols than as an art of organizing their thoughts"
- Central theme: concept creation and abstraction creation is fundamental in CS and the standard undergraduate math curriculum may not provide any of that training

CS = Mastering Complexity

- Minsky, 1979: “In many ways, the modern theory of computation is the longawaited science of the relations between parts and wholes; that is, of the ways in which local properties of things and processes interact to create global structures and behaviors.”
- Minsky, 1979: “Like mathematics, [computer science] forces itself on other areas, yet it has a life of its own. In my view, computer science is an almost entirely new subject, which may grow as large as physics and mathematics combined.”
- Djikstra: As a consequence of the hierarchical nature of computer systems, programmers gain agility with which they “switch back and forth” between “various semantics levels”, “between local and global considerations”, and between “microscopic and macroscopic concerns”. He regarded this as an extraordinary ability among scientists.
- Djikstra, 1987: “The ratio between an hour (for a while computation) and several hundred nanoseconds (for an individual instruction) is 10^{10}, a ratio that nowhere else has to be bridged by a single science, discipline, or technology.”

CS is its own thing

- Dijkstra, 1997: “Another thing we can learn from the past is the failure of characterizations like ‘CS is really nothing but X’, where X is your favorite discipline, such as numerical analysis, electrical engineering, automata theory, queuing theory, lambda calculus, discrete mathematics, etc.”

Theory/Practice War

- Against the lofty aspirations of Dijkstra and Knuth, the business world claimed that academic computer science had detached from the needs of the real world
  - 1972, CACM: “Industry gets graduates from computer science departments with a bag full of the latest technical jargon but no depth of understanding the real computer systems and no concepts of the problems they will be asked to solve
  - 1975, Educom: “A formal education in computer science is not an adequate background for those who must design and install large-scale computer systems
- Punchline: CS at the time was accused of not being able to answer the “software crisis”

“Software Engineering”

- Term introduced in 1968 at a conference held to discuss the software crisis
- Introduced as a “practical subject”
  - Concerned with building systems, and in this sense definitely “engineering”
  - Software engineer should be a jack-of-all-trades
- Radically opposed to Dijkstra’s view
  - He called software engineering “The Doomed Discipline” and defined it as “how to program if you cannot”
  - If your goal is to make CS accepted as a noble science, you can’t accept that part of it is “just” engineering
- But, right before the 1980s, software engineering had become established as part of CS

Yet Another Definition

- 1978, ACM SIGCSE: “Computer Science is the study of the theory and practice of programming computers. This differs from the most widely used definition by emphasizing programming as the central notion and algorithms as the main theoretical notion supporting programming”
- Message: Programming is central, and the theory is useful only as far as it supports programming

’68 vs. ’78 ACM curriculum

- 1968
  - information structures and processes
  - information processing systems
  - methodologies
- 1978
  - programming topics
  - software organization
  - hardware organization
  - data structures and file processing
- No “theory” in the 1978 curriculum!!!!
- The 1978 curriculum also emphasized “hands-on” work in its description
- In 10 years, theory ⇨ programming and applications
- Answer to the “software crisis” provided

Some Backlash

- The ’78 report was highly criticized for lacking mathematics/theory and implying “computer science = programming”
- ’68 document: “an academic program in CS must be well based in mathematics since CS draws so heavily upon mathematical ideas and methods”
- ’78 document: “no mathematical background beyond the ability to perform simple algebraic manipulation is a prerequisite.... mathematics is not required as a prereq for any hat the core curriculum”
- The following was added to the ACM ’78 document: “An understanding of and the capability to use a number of mathematical concepts and techniques are vitally important for computer scientists.”

The 70’s and 80’s

- Expansion of scope and interdisciplinarity
  - Computers are no longer just calculators, but can partake in society
- Didn’t make the discipline easier to define :)
- We’re seeing the effects of that development today…
- 1979, Minsky: “CS has such intimate relations with so many other subjects that it is hard to see it as a thing in itself”

Denning's Definition

- In 1985, in American Scientist, Peter Denning defined computation science as: "The body of knowledge dealing with the design, analysis, implementation, efficiency, and application of processes that transform information"

- I happen to really like this definition, and so did the ACM...

Recent Considerations

- 1989, ACM/IEEE task force comes up with Definition: “[CS is] the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, “What can be (efficiently) automated?”
- Identifies 3 major traditions in CS
  - Theory: mathematics
  - Abstraction (modeling): natural sciences
  - Design: Engineering
- The report states that debating whether one above is more important than the others is likely counter-productive. They are so intricately intertwined that it is irrational to say that one is fundamental
- Brooks around 1996 addresses this...

Brooks' 4 unhappy trends in CS

- (1) Accepting a pecking order that theory is more respectable than practice
- (2) regarding the invention and publication of endless varieties of computers, algorithms and languages as the end
- (3) forgetting the users and their real problems (academic ivory tower)
- (4) directing young and brilliant minds towards theoretical subjects at the expense of more applied subjects

CS and Cars

- Kugel, 1988: “Lots of people drive cars but that does not justify an ‘automotive science’. Do computers justify a computer science?”

- Smith, 1998: “Computers turn out in the end to be rather like cars: objects of inestimable social and political and economic and personal importance, but not the focus of enduring scientific or intellectual inquiry.”

- Dijkstra would answer that the computer is to the computer scientist what the telescope is to the astronomer (and yet, we don’t have “telescope science”)

Outline

- History of the identity of the field
  - Funny / illuminating quotes abound
- Discussion of the three CS traditions
  - math, science, engineering
- Fundamental question

The 3 traditions

- This has been a rocky history
  - theory vs. practice, computer vs. computing
- It seems that accepting CS as 3 traditions is the key to inner peace
  - mathematics, engineering, natural science
- Some areas make research contributions across traditions, some don’t
- Let’s looks at some arguments for/against each tradition

Is CS Mathematics?

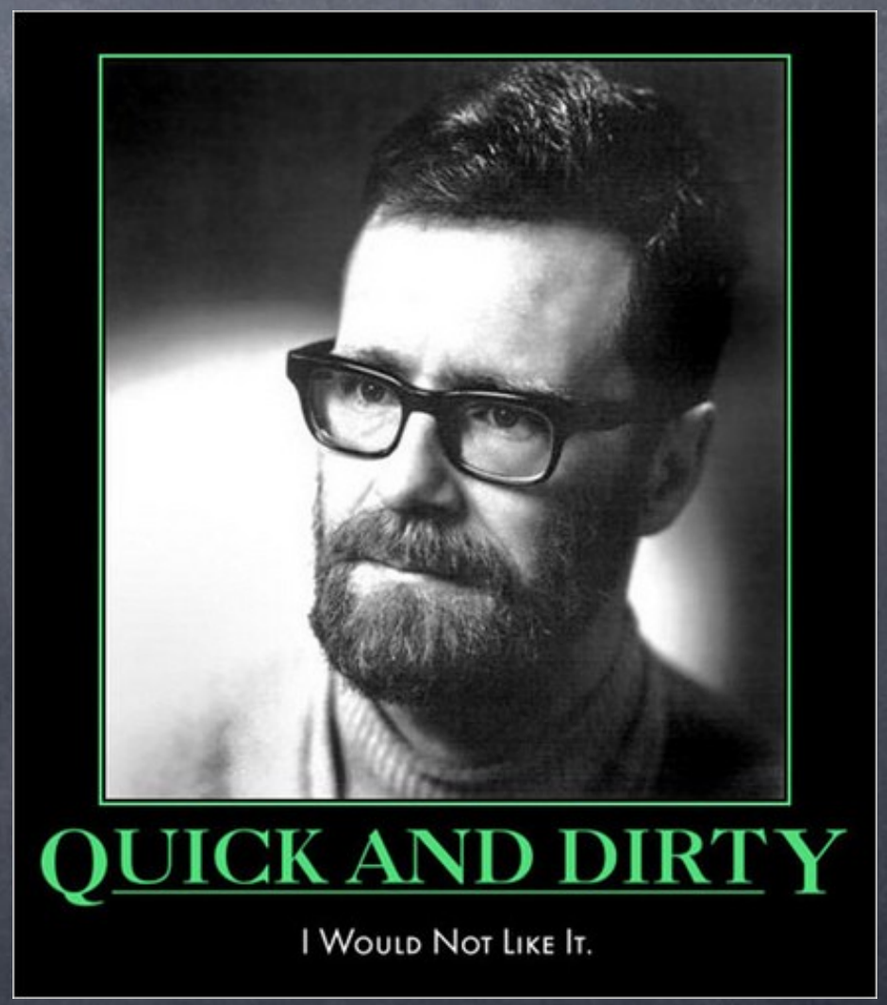
- Most radical positive view: Reductionism (a discipline can be fully explained by another)
  - Reductionist's view of chemistry: it's only physics
  - Reductionist's view of psychology: it's only biology
  - Reductionist's view of CS: it's only mathematics!
- Many of the early heroes of CS were mathematicians
- Most impressive theoretical advancements in CS are always always proven and formulated in the language of mathematics
- So there is a math basis, but a full reduction to math???
- Typical argument: programming is not math
  - And yes, we say that programming is part of CS

Is Programming Math?

- Hoare, 1969
  1. Computers are mathematical machines. Every aspect of their behavior can be defined with mathematical precision, and every detail can be deduced from this definition with mathematical certainty by the laws of pure logic.
  2. Computer programs are mathematical expressions. They describe with unprecedented precision and in every minutest detail the behavior, intended or unintended, of the computer on which they are executed.
  3. A programming language is a mathematical theory. It includes concepts, notations, definitions, axioms and theorems, which help a programmer to develop a program which meets its specification, and to prove that it does so.
  4. Programming is a mathematical activity. Like other branches of applied mathematics and engineering, its successful practice requires determined and meticulous application of traditional methods of mathematical understanding and proof.

Is Programming Math?

- Hoare “complained” that because computers and programs are not typically constructed with mathematical rigor, we’re forced to use experimentation
  - But we should aspire for more!
- Dijkstra’s view is similar: CS is the study of certain mathematical expressions: algorithms
- Formal derivation from specification to algorithm by a sequence of



Formal Verification

- Formal verification of programs: going from program to specification
  - Generates proofs that programs are correct
  - Reversed Dijkstra/Hoare
- Today, formal methods are widely studied, but we're all aware of the limitations
- In the 60's/70's, that seemed to be the way of the future for many

- The debate between the "it's math" and the "it's not math" camp was centered around formal verification (i.e., are programs expressions to be proven?)

Objection #1: Proofs

- Proofs in CS ≠ Proofs in Math
  - De Millo et al., 1976: “no mathematicians grasps a proof, sits back, and sighs happily at the knowledge that the theorem is true”
    - they run out of the office, show the proof to excited colleagues, and then maybe publish after modifications/fixes
  - De Millo et al., 1976: “The verification of even a puny program can run into dozens of pages, and there's not a light moment or a spark of wit on any of those pages. Nobody is going to run into a friend's office with a program verification. Nobody is going to sketch a verification out on a paper napkin. Nobody is going to buttonhole a colleague into listening to a verification. Nobody is ever going to read it. One can feel one's eyes glaze over at the very thought.”
- Granted, but it’s not because it’s tedious and automatically generated that it’s necessary worthless

Objection #2: Real World

- Fetzer, a philosopher, in 1998 states that a program cannot be proven correct because it operates in a concrete world
  - Criticism of Hoare's statement (1969): "When the correctness of a program, its compiler, and the hardware of the computer have all been established with mathematical certainty, it will be possible to place great reliance on the results of the program, and predict their properties with a confidence limited only by the reliability of electronics."
- Fetzer distinguishes: programs-as-text (algorithms) and programs-as-causes (executable)
- He argues that mathematical certainty is not possible for the latter due to external events
  - We can prove algorithms corrects, but not programs
- This is a philosophical argument and for instance when we write a research paper we make tons of implicit assumptions that the real world behaves "ok" (e.g., no cosmic rays)
- Received a HUGE response, or rather backlash

Objection #2: Real World

- In 1989, CACM article co-authored by 10 well-known computer scientists, stating that Fetzer's position was "ill-informed, irresponsible, and dangerous"
- Fetzer's answer: "In its inexcusable intolerance and insufferable self-righteousness, [their] letter exemplifies the attitudes and behavior expected from religious zealots and ideological fanatics, whose degrees of conviction invariably exceeds the strength of their evidence."
- Answers/Responses went on for a while
  - To the point that Fetzer wrote the "story" of all this
- Many failed to see the difference between program-as-text and program-as-causes
- Bu as long as one recognizes this difference, one still has a strong argument for formal methods for the former
  - Especially if CS is not about executing program, but about studying computation

Objection #3: Users

- Hoare, 1969: “The most important property of a program is whether it accomplishes the intentions of its user. If these intentions can be described rigorously by making assertions about the values of variables at the end (or at intermediate points) of the execution of the program, then the techniques described in this paper may be used to prove the correctness of the program.”
- Only sensible if program computes formal functions
- Proving a program correct does not mean that the program does what the user wants it to do
- Today, proponents of formal methods (i.e., mathematical basis for CS) still see value in empirical methods
  - Formal methods do not make testing unnecessary
  - But using them can lead to much better designs
- Bowen et al., 2005: “The software engineering community is not willing to abandon formal methods [...] but neither is it willing to embrace them”

Is CS Engineering?

- Many early CS pioneers were engineers
  - Some CS department are in engineering colleges
  - Some CS departments are combined with CE departments
- If the goal of CS is to build useful things, then it is engineering
- Commonly drawn line: Computer engineers work with physical things (hardware), and computer scientists work with abstract things (algorithms)
- But turns out it’s really hard to keep that line not blurred
- Engineering research: development of “tools” that will enable classes of tasks to be accomplished more efficiently
  - Sounds a lot like CS research
- Many have argued that CS is “synthetic engineering”

is CS Engineering?

- The engineer compare solutions and select solutions in terms of costs and efficiency
- Note that a lot of “theoretical” CS is just about this: algorithm complexity is nothing but time/storage cost!
  - Could it be that even theoretical CS is research???
- Some may argue that the goal of these works is to discover algorithm principles and properties of computation
  - But many articles are couched as “their optimal algorithm was $O(n^x)$ and ours is $O(n^{y<x} \log n)$”
- And if CS is a huge field today with many departments, an industry, etc. has a lot to do with the fact that the ENIAC was built
  - Instead of remaining profound/amazing/incredible/etc. but idle speculation by Turing/Church

Objections

- Objections abound about “software engineering”
- You can imagine what Hoare/Dijkstra/etc. would say
  - Main criticism: software engineering lacks rigor because it’s “just” about building something that works
- Holoway, 1995: software engineering is based on “a combination of anecdotal evidence and human authority”
- Zelkowitz, 1997: about 1/3 of 600 surveyed software engineering articles had no experimental validation
- Question: can engineering contributed anything to the common knowledge about computing? And if not, should it be part of the academic discipline of computing?
  - Not all important activities must be nominated as academic disciplines

Objections

- Consider a result like: Here is the first polynomial algorithm to solve problem P. Its complexity is $40^{60}n^{12} + O(n^{11})$
- The engineer: no use whatsoever
  - We can never use it to build anything
- The mathematician: huge result because P is not NP-hard
  - We've learn something about computing

Is CS a Science?

- This question has been debated a lot
  - CS is math, and so it’s not (natural) science!
  - CS is engineering, and so it’s not science!
- This is a very complex question
  - The term “science” is the subject of entire books that try to explain what it is
- D. Knuth: “unnatural science”

Human-made Argument

- Common objection: science is about understanding the world around us, and CS deals with human constructions
  - New CS results do not tell us anything new about the world, only about how well previous computer scientists have done their job!
  - A human had to think up problems to solve
    - In fact, it’s all about defining problems!
- “In CS there is no history of critical experiments that decide between the validity of various theories, as there are in physical sciences” [Hartmanis, 1993]
  - I TOTALLY disagree with this (e.g., network models)
- Advancements of natural sciences are often documented by dramatic experiments, in CS they are often documented by dramatic demonstrations [Hartmanis, 1993]
- The sciences are focused on what exists, CS focuses on what can exist [Hartmanis, 1993]

Human-made Argument

- Some argue that this is a red herring
  - Tichly 1998: “the only major difference between traditional sciences and CS is just that information is neither energy nor matter”
  - What defines a science is not “what” is being studied, but “how” it is studied

Empirical Argument

- Ok, it's not "natural" but it still has to be "empirical", so who cares?
  - But for the most "mathematical" part of it

The scientific method	***Problem-solving in computer science***
1. Formulate a *hypothesis* for explaining a phenomenon	1. Formulate an *algorithm* for solving a problem
2. *Test* the hypothesis by conducting an *experiment*	2. *Test* the algorithm by writing and running a *program*
3. *Confirm* or *disconfirm* the hypothesis by evaluating the results of the experiment	3. *Accept* or *reject* the algorithm by evaluating the results of running the program

- We need this because of the gap between models in CS (e.g., a program's view of the world) and the real world
  - Showing that the program does what it's supposed to be doing is just not enough
  - And often programs try to do things we don't even understand (e.g., AI stuff) ("we're better at building systems than at understanding

Empirical CS Research

- There is thus a strong need for empirical CS Research
  - And of course, for all the cross-disciplinary fields (e.g., HCI) as well
- A problem is that we don't do it well!
  - Think of the gap between CS courses and CS research
- There are many studies that compare CS research to research in other fields, and show that CS researchers experiment significantly less, even though they should
  - But we can fix this!
  - Some of course say "but we're not studying the same things, so why should we use the same methods?"

Empirical CS

- So in the end, whether CS is a science or not isn’t perhaps very interesting
- But many CS research activities have to rely on empirical scientific methods
  - Which is not always done well due to our training (too much emphasis on the mathematical and engineering aspects of CS?)

Outline

- History of the identity of the field
  - Funny / illuminating quotes abound
- Discussion of the three CS traditions
  - math, science, engineering
- Fundamental question

Fundamental Question

- When’s somebody asks “What’s CS?”
- It would be nice to have an answer
- One option is to have them study the philosophy of CS
- But it would be nice to say “Well, the fundamental question that the discipline answers is....”

The Fundamental Question

- Forsythe, 1969: “The question ‘What can be automated?’ is one of the most inspiring philosophical and practical questions of contemporary civilization”
- Denning, 1989: “Fundamental question: ‘What can be (efficiently) automated?’”
  - “Effective”: the steps of an algorithm must be exact and each step must be executable in finite time (Knuth, 1997)
  - “Efficient”: statement about the behavior of an algorithm with different sized input (Knuth, 1997)
- Considering human-centered computing (since the 80‘s), the question should be “What should be (efficiently) automated?”
  - What will help users? What’s ethical?

Theory vs. Practice

- “What can be automated?”
  - This is a theoretician’s question
  - Church-Turing thesis divides processes into those that can and those that cannot
  - Leads to other theoretical questions:
    - “Why can this class of processes be efficiently automated?”
    - “What kinds of properties are typical of processes that can be efficiently automated?”
- “How can process p be automated?”
  - This is a practitioner’s question, with many implied considerations (cost, maintainability, etc.)
  - Leads to other practical questions:
    - “Which implementation automates p most efficiently?”
    - “What kinds of problems do certain implementations automate best?”

Rewording

- Taking into account both theory and practice, and considering that computing is human-centered, Matti Tedre rewrites the fundamental question of the field as: “How can one efficiently and reliably automate processes that can be automated and that should be automated?”

Fundamental Question

- Note that this fundamental question removes the “scientific” aspect
  - Consider a paper that studies power-laws of vertex degree in Internet topology
  - Consider a paper that studies failure rates in servers and hard drives
- Such works are not at all about computing itself but they study human-made (computing) systems the way a biologist would study organisms
- But not all CS works have to study the fundamental question...
  - There is no way there will be one catch-all question anyway

Conclusion

Conclusion

Conclusion

- If anything, we've learned about
  - how the field developed
  - the perspective of "heroes" of olde
  - how the field if polymorphic and not "one" thing
- Hopefully, some of you will be motivated to read further on this topic
  - If everybody was very knowledgeable in the philosophy of CS, we'd be in a better world