A Look at the Origins of Computing's Disciplinary Ways of Thinking and Practicing

Paris, May 18, 2017

Matti Tedre
University of Eastern Finland

This Talk

1. Computational thinking: an introduction
2. Many roots of CT
3. Making of computing education
4. Computational thinking for everyone
5. Computational thinking in 2017
6. Summary and opportunities

Angle to questions

How do we analyze computational thinking from the perspectives of the history and philosophy of science?

Computational thinking: An Introduction

5/26/17

Computational Thinking

An attempt to capture

computing's disciplinary ways of thinking and practicing

Computational Thinking

A buzzword since 2006

- Lavish funding
- Journal issues (IJCCI, TDJ, …)
- Research centers (CMU, …)
- STEM curriculum elements (CSTA, CAS,..)
- Books

Why Computational Thinking?

Computational Thinking …

"is a fundamental skill for everyone"

—Wing (2006)

Computational Thinking…

… "teaches everyone to succeed in the new digital world"

—EdSurge (2016)

Computational Thinking…

… "benefits society"

—Wing (2014)

Computational Thinking…

… "is a fundamental intellectual skill"

—Linn et al. (2010)

Computational Thinking…

"is a universally applicable attitude and skill set"

—Wing (2006)

Computational Thinking…

"[CT] performance … can predict the student's future academic success"

—Haddad & Kalaani (2015)

Computational Thinking…

"improves analytical skills of majors as well as non-majors"

—Van Dyne & Braun (2014)

Computational Thinking…

"[Correlates with student's] academic performance in a primary school"

—Oliveira, Nicoletti & del val Cura (2014)

Computational Thinking…

is "careful reasoning about the methods of doing things"

—Sussman (2010)

Computational Thinking…

"is comparable to reading, writing, **speaking**, and arithmetic"

—Linn et al. (2010)

What is Computational Thinking?

"computational ideas and habits of mind that people in computing disciplines acquire through their work in designing programs, software, simulations, and computations performed by machinery"

Tedre & Denning

What is Computational Thinking?

In pedagogical terms "**computing's disciplinary ways of thinking and practicing**"

When was CT born?

"a term coined by Jeannette Wing" (Lee et al., 2011)

"Wing [12] launched a discussion regarding the role of CT across all disciplines" (Barr & Stephenson, 2011)

"defined first by Wing"… "seminal paper"… "pioneering article"

Many Roots of Computational Thinking

Late-1600s

1700s

1700s

1800s

1800s

1800s

Late 1800s

Early 1900s

1950s: Engineering and design

- CT detaches from human computing
- Drivers of computing practice — (ways of practicing)
  - "How" questions, practical aims
  - Efficiency, control, usefulness
  - Solution-driven
  - Demonstrations as milestones
- Debugging

1950s: Theory for technology

- Connections between computing machinery and mathematical logic
  - Turing machines

50s/60s: Programming methodology

- Programming methodology & programming language concepts
  - Abstract data types
  - Encapsulation
  - Information hiding
  - Structured programming
  - Recursion [cf. "definition by induction"]
  - Complexity of computation
  - Rigorous analysis of programs

Making of Computing Education

Establishing disciplinary ways of thinking and practicing

1940s: Numerical methods programs

- Grosch, Aiken, Eckert
  - 1946–48 new educational programs for the use and design of large scale machine computing and numerical methods
  - Design + construction of machinery
  - Tabulating machinery etc.

Early 1950s: Training for technical jobs

- Computer mass-production starts
- New kinds of jobs but no people to do them
  - Programmers, operators, technicians
  - 1954 "*First Conference on Training Personnel for the Computing Machine Field*"
  - 1954 US training capacity: 260 people / yr
- Most people trained by companies
  - Professional identities unclear

1950s: Pulling the pieces together

- Build many things from scratch:
  - Curricula (for new job descriptions)
  - Staff training, facilities
  - Textbooks
  - Course material and examinations
  - Mental models for thinking about programming
- Theoretical base unclear
- Unrealistic expectations: 2–12 weeks

Late 1950s: Training for applications

- IBM mass donations of computers to universities
  - Applied topics—business forecasting, industrial analysis, use of electronic data processing
  - Mathematical elements
- No consensus over curriculum
  - Based on the educators' interest and focus
  - Educators often not university staff
  - University computing centers

Early 1960s: Training for software development

- Major organizations run curriculum efforts
  - DPMA, IFIP, IEEE, ACM, etc.
- Software industry 10x larger every decade
  - Most jobs in data processing (applied)
- Educational complaints at the time:
  - Underdeveloped pedagogy
  - Lack of up-to-date textbooks
  - Faculty availability
  - Graduates not ready for work

Early 1960s: Computing's unique thought processes

- "Algorithmizing"
  - Computing for all fields of science
  - Liberal arts colleges
- "General-purpose thinking tools"

Late 1960s: Standardization

- Number of computing majors doubles every 15 months
  - Still increased pressure from the industry
- ACM CC'68 gets adopted in universities
  - SIGCSE 1969
- Industry likes also DPMA, IFIP, etc.
  - More practical orientation
  - Software engineering movement — 1968

Early 1970s: Training for the academia

- Communities of practice arise in CSE
  - SIGCSE, BCS, DPMA, …
  - Discussions: programming paradigms, languages, software engineering, structured programming, pedagogy, professionalism, law, social relevance
- How to teach computing as an academically respectable discipline?
  - Atmosphere set by formal verification

Early 1970s: Training for the academia

- Computing courses built on the field's own "first principles"
  - Systems courses, programming, algorithm courses
- Strong appropriation of concepts into computing concepts [since 1950s]
  - Graphs, matrices, strings, etc.
  - These could be taught in terms of *computing*, not in terms of mathematics

The elusive disciplinary agenda?

- Captured in
  - Magazines (Datamation, CACM, Data-Link, …)
  - Popular books, textbooks, essays
  - Professional organizations (DPMA, IFIP, …)
  - Influential research groups, pioneers
  - Software practice
  - Curricula (esp. CC'68)
- Computational thinking (?)

Ways of thinking and practicing

- Captured in
  - Perlis 1959: "Algorithmizing"
  - Dijkstra 1974: "Algorithmic thinking"
  - Knuth 1974: Computing's "types of thinking"
  - Sheil 1980: "Procedural literacy"

Computational thinking for everyone

"We were, in part, evangelists with a message about computing machinery. The gospel is certainly widespread [today]"
Householder (1955)

The computational thinking gospel

- For sciences ('50s)
- For liberal arts colleges ('60s)
- For K–12 education ('70s)
  - The "transfer hypothesis": learning programming makes you a better thinker in many other domains of knowledge, too.
  - One of the most persistent myths in computing

The transfer hypothesis [1960]

The transfer hypothesis [1968]

The transfer hypothesis [1969]

The transfer hypothesis [1971]

The transfer hypothesis [1974]

The transfer hypothesis [1970]

Responses from the Education Community

No evidence of supporting mathematical rigor, exploration in learning, or transfer of skills from programming

No evidence of transfer

5/26/17

Very rarely and only when you explicitly plan for it

5/26/17

No evidence of connection beyond anecdotal and introspective evidence

5/26/17

Transfer Hypothesis Carries On

The transfer hypothesis
[1986]

5/26/17

The transfer hypothesis
[1993]

5/26/17

The transfer hypothesis
[2006 onward]

5/26/17

Computational Thinking Status update: 2017

Status update: 2017

+ Well funded
+ Popular
+ Government-supported
- Dozens of definitions
- Ahistorical
- Exaggerated
- Narrow

Challenge 1: Keeping the ambition alive

- Previous visions: A radically different way of looking at the world
  - Epistemological revolution [Beynon, diSessa, …]
  - Pedagogical revolution [Papert, Minsky, etc …]
  - Scientific revolution [Wilson, Baltimore, etc …]

Challenge 2: Avoiding dogmatism

- Some previous visions:
  - Epistemological pluralism [Papert, Turkle, etc …]
  - Alternative ways of knowing [Bolter, …]
  - Not superior to many other ways of thinking
- "Computational chauvinism" (Tedre, Denning & Yongpradit)

Challenge 3: Direction of fit

- **The driver of programming skill** or **the outcome of programming practice?**

Challenge 4: Exaggerated claims

- The transfer problem
- Exaggeration risks us as oversellers of CT

Challenge 5: Avoid narrow views of computing

- "CS=programming"?
- "Coding"?!?

Challenge 6: Question the role of formulation

Challenge 6: Question the role of formulation

- Let's be precise: We don't formulate problems!
- We write software to meet a concern, answer a threat, rise to an opportunity
- We may formulate **problem statements**…
  …but it's much more complex in practice.

Challenge 7: Don't lose computational models

- "CT is about how humans think" … [2008]
- …But how much is it about how we think as opposed to how we can control computers?
- The purpose of programs is to control machines

Don't lose sight of computational models

- Hurricane Sandy example (Wang, 20177)
  - "With CT at multiple levels, dare we say that many of the disasters from Sandy might be substantially reduced"
  - "What if streets are flooded? What if flood water enters the subway?"
  - "Should we waterproof generators in designated at-risk buildings?"
  - "Do we have firefighters trained for boats?"

Don't lose sight of computational models

- Wang, 2017:
  - *CT: Small Is Beautiful*
    Reduce the camera resolution when taking pictures to share online.
  - *CT: Google It*
    The answer is "Google."
  - *CT: Be Careful with Online Information*
    Be critical; don't believe everything online. Avoid spreading untruth.

Summary

Summary

- CT is a wrapping for computing's driving ideas
  - Powerful mental tools for designing computations
  - Enriches science through new perspectives and tools
- Has become very popular in the past 5 years

Opportunities

- CT lacks historical depth
  - Different from disciplinary history
  - Scattered histories of ideas
- Practical problems with ahistoricity
  - …Dead-ends frequently visited
  - …Exaggerations repeatedly debunked
  - …Powerful ideas forgotten and "rediscovered" or rebranded

Opportunities

- CT lacks philosophical depth
  - Conceptual problems abound
  - Epistemologically shallow
  - Ignores central lessons from the philosophy of computer science

Opportunities

- CT could benefit from analysis from a philosophy of science perspective
  - Disciplinary ways of thinking and practicing (Education)
  - Mangle of practice? (Hacking)
  - Research agenda? (Mahoney)
  - Hard core of computing's research programme? (Lakatos)
  - Disciplinary matrix? (Kuhn)