Towards a History of Model-Modellee Conflations in Computer Science

Edgar G. Daylight — Siegen University — 8 Feb. 2018

*Lille Workshop:*
Models betw. structures and meanings of programs

“Engineers who acknowledge limits to technology are all too easily cast as pessimists or failures.”

[Slayton, 2013, p.11]
Outline

1. A Bird’s Eye View
2. A Worm’s Eye View
3. Intermezzo
4. My Personal Contribution
5. Terminology & Methodology
A Bird’s Eye View
What Can We Learn from History?
ARGUMENTS THAT COUNT

PHYSICS, COMPUTING, AND MISSILE DEFENSE, 1949–2012

REBECCA SLAYTON
“By the time physicists began to note the limitations of software, the missile defense program was moving forward with a momentum all its own.”

[Slayton, 2013, p.84]
< missing picture about human-machine symbiosis >
Does History Repeat Itself?
“Historians assess past examples of change to better understand change in society today.”

[Stearns, 1998]
“Dozens of demonstrations by hackers and security researchers have proven it is entirely possible for criminals 1500 miles away to seize control of your car when you are driving 65 miles per hour down the highway.”

[Goodman, 2016, p.363]
Governments and industries simply do not want to “miss the bandwagon of tests, first deployments, and perhaps manufacturing too.”

[De Grave, 2016]
Perhaps There is No **Technological Fix**
In the *Communications of the ACM*:  

“A steady march toward the automated car is clearly under way”  

[Casner et al., 2016]
Can History Help to Prevent Mistakes?
A historical approach is needed that renders errors visible and gives us a broader, more contextual view of software and its actual and potential development, and the implications this has for science and society.

Without knowledge of the past, there is no future.
Can Historians Help to Prevent Mistakes?

— Yes we can!
A Worm’s Eye View
< missing pictures of Ronald Reagan and the Star Wars project, available on the Internet >
Slayton’s Narrative Suggests that There is No Technological Fix for Missile Defense
“Most of the many published proofs of programs [in computer science] are actually proofs about models of programs, models that ignore the very properties of digital computers that cause many of the ‘bugs’ we are trying to eliminate.”

[Parnas, 2012]
Begin of Intermezzo
Computer scientists, have an “inclination to slide, in their discourse, between programs, models, and theories as if there was no distinction between them.”

[Moor, 1978]
Computer scientists — and relatively few software engineers and security experts — believe in a Turing Fix in that they, often heedlessly, treat a laptop as a Turing machine or a computer program as a Turing machine program in their research, thereby downplaying the modelling activity at hand.
Conflations in various fields:

**Linguistics**: mistaking sentences deduced from a formal grammar for a natural language [Schüttpelz, 1996, Ch.3].

**Cognitive studies**: equating the computation of functions with actual cognition [Fetzer, 1998, p.378].
Conflations in various fields:

**Computability:** confusing a Turing machine with a stored-program computer [Daylight, 2014].

**Software engineering:** mistaking a C computer program for a system realization [Lee, 2015, p.4839].
Conflations in various fields:

**Database management:** “representations and reality are conflated” by computer scientists, “it is difficult to comprehend privacy in any terms other than control over access to *individually identified* data records” [Agre, 1997].
Conflations in various fields:

**Database management:** “representations and reality are conflated” by computer scientists, “it is difficult to comprehend privacy in any terms other than control over access to *individually identified* data records” [Agre, 1997].

**Formal verification:** confusing verification of mathematical models with truth [De Millo et al, 1979].
“Models often have properties real mechanisms do not have, and it is possible to verify the correctness of a model of a program even if the actual program will fail.”

[Parnas, 2012]
End of Intermezzo
Is There a Technological Fix for Internet-Connected Pacemakers?
< missing cartoon about pacemakers, available on the Internet >
Stakeholders who (implicitly) believe in a foolproof mapping — that is, who *conflate* the security model of the pacemaker’s software with the deployed software itself (= modellee) — will put too much weight on the “security proofs” obtained by researchers.
What is missing in the current state of the art is ...
... an in-depth history of software and its mathematical underpinnings
Too Much Technical Jargon for Policymakers
Error-free software is a necessary (but not a sufficient) condition for having a secure — and, hence, a humanly safe — Internet-connected, cyber-physical system.

www.dijkstrascry.com/Lee1
OVERARCHING RESEARCH QUESTION

How did software specialists technically cope with the problem of having software errors (in deployed systems) during the course of history, from the 1950s till today?
Answering this Research Question is Urgent
My Personal Contribution
(forthcoming)
Fill two research gaps:

1. **Computer science** history *pur sang*

2. The socio-technical history of **software engineering**
For Computer Scientists, the Role of Mathematical Logic is that of a Queen

Courtesy of Michael A. Jackson
For Software Engineers, the Role of Mathematical Logic is that of a Humble Servant

Courtesy of Michael A. Jackson
Understanding technical developments of software engineering for drones, cars, and pacemakers requires comprehension of computer science’s decades-old appropriation of ideas from mathematical logic in their continual move away from the real, physical world.
After all, logico-mathematical rigor in computer science has led to the development of software analysis tools that engineers use daily.
The Practice of Conflating is Both Powerful & Troubling
1) Mathematical rigor in the science of computer programming ...

2) Ignorance of the model’s limited regime of applicability ...
< missing picture about human-machine symbiosis >
Terminology & Methodology
Computing vs. Computer Science
Computing vs. Computer Science

The **history of computing** includes the history of the abacus, punched card machines, personal computers, computer games, the computing industry, and so forth.
Computing vs. Computer Science

The **history of computing** includes the history of the abacus, punched card machines, personal computers, computer games, the computing industry, and so forth.

While these kinds of histories have been published, computing history’s modern chapter on **computer science** has yet to be drafted, let alone written.
Computer Science vs. Software Engineering
Computer Science vs. Software Engineering
Computer Science vs. Software Engineering

• The *Scientific Mechanism* amounts to choosing a model that is faithful to the modellee.
Computer Science vs. Software Engineering

• The **Scientific Mechanism** amounts to choosing a model that is faithful to the modellee. A scientist studies nature — and a computer scientist studies the nature of the artificial (e.g. computers) — and subsequently chooses or invents a model that s/he deems is useful.
Computer Science vs. Software Engineering

- The *Scientific Mechanism* amounts to choosing a model that is faithful to the modellee. A scientist studies nature — and a computer scientist studies the nature of the artificial (e.g. computers) — and subsequently chooses or invents a model that s/he deems is useful.
- E.g., the laws of classical mechanics (= model) are an attempt to capture and predict motions in our physical world (= modellee).
Computer Science vs. Software Engineering

• The **Scientific Mechanism** amounts to choosing a model that is faithful to the modellee. A scientist studies nature — and a computer scientist studies the nature of the artificial (e.g. computers) — and subsequently chooses or invents a model that s/he deems is useful.
  • E.g., the laws of classical mechanics (= model) are an attempt to capture and predict motions in our physical world (= modellee).

• The **Engineering Mechanism** amounts to producing a modellee that is faithful to the model.
  • E.g.: a software engineer uses a flowchart (= model) to produce a program (= modellee).
The Symmetry Principle
The Symmetry Principle

“I am simply attending with equal curiosity to both sides of the dispute and reporting how the historical actors saw one another.”

[Bloor, 2014, p.404]
The Symmetry Principle (Bycroft, 2016)

1. Should not assume in advance of empirical inquiry that:
   a. true beliefs are best explained rationally
   b. false beliefs are best explained irrationally

2. Says nothing against the practice of internal history of science
Thank You!
Thank You!

“Engineers who acknowledge limits to technology are all too easily cast as pessimists or failures.”

[Slayton, 2013, p.11]
Sources

- [HoC, 2010s] Reference to the following sources on the History of Computing:
Sources

Sources


• [The Economist, 2017] Two entries in The Economist:
