# The Computer Science Undone: How The Social Construction of Disciplinary Boundaries and Disciplinary Hierarchies Shape a Field

*Felienne Hermans*
Vrije Universiteit Amsterdam (The Netherlands)

**Résumé :** Quels sont les principaux thèmes de la science informatique et comment sont-ils étudiés ? Ces questions sont centrales en philosophie de l'informatique. Cependant, les réponses à ces questions font débat et dépendent de constructions sociales. Cet article s'inspire des questions auxquelles Shirley Gregor a répondu dans son travail fondateur sur la façon dont la théorie est utilisée dans le domaine des systèmes d'information. Cet article vise à mieux comprendre comment ces mêmes questions sont abordées dans le champ de la philosophie de l'informatique. En particulier, nous analysons quatre livres récents qui traitent des thèmes et méthodes de l'informatique. Bien que chacun de ces livres ait sa propre perspective et utilise une approche différente, ils arrivent tous à des conclusions similaires : l'informatique est avant tout l'étude du calcul, des algorithmes et de leur implémentation et vérification. En termes de méthodes employées, les quatre livres s'accordent sur trois traditions distinctes : une tradition mathématique, une tradition d'ingénierie et une tradition scientifique empirique. Nous constatons que les frontières disciplinaires de l'informatique sont retracées dans chaque livre en mettant l'accent sur des travaux tantôt mathématiques, tantôt qualitatifs ou encore empiriques plutôt que sur d'autres formes de savoir. Cet article vise aussi à poser la question de la part de l'informatique qui est non-faite. Quelles méthodes de recherche n'utilisons-nous pas à cause du focus actuel sur les façons de penser propres aux mathématiques, à l'ingénierie ou aux sciences ? Quelles questions de recherche sont hors du champ de l'informatique d'aujourd'hui ? En tant qu'informaticiens, avons-nous suffisamment d'influence sur le monde des ordinateurs et au-delà ?

---

*Philosophia Scientiæ*, 30(2), 2026, 3–30.

**Abstract:** What are the core topics of Computer Science, and how are these topics studied? Those are the central questions of the Philosophy of Computer Science. However, how those questions are answered is a matter of ongoing debate and social construction. Guided by the questions answered by Shirley Gregor in her seminal work on theory in the field of Information Systems, this paper aims to understand how contemporary work on the philosophy of Computer Science answers these questions. In particular, we examine four recent books explicitly examining the topics and methodologies of Computer Science. While each of the books takes a different perspective and uses a different approach, all of them reach similar conclusions: Computer Science is firstly the study of computation, of algorithms and their implementation and verification. In terms of methods employed, the four books agree there are three separate traditions: a mathematical tradition, an engineering tradition and a empirical scientific tradition. We see that the disciplinary boundaries of Computer Science are redrawn in each of the books, emphasizing mathematical, qualitative and empirical work over other forms of knowing. This paper also aims to ask the question what the Computer Science is that is currently *undone*. What research methods aren't we using due to the focus on mathematical, engineering or scientific reasoning? What research questions fall outside of the scope of current Computer Science? And are we as computer scientists influencing the world of computers and beyond enough?

# 1 Introduction

I consider myself an academic nomad in Computer Science. I started my career in Theory, then worked in Programming Languages and Software Engineering, in Computer Science Education, and then again in Programming Languages. Two decades of working in different subfields[1] have allowed me to reflect on the views and practices of Computer Science as a whole. In my experience, Computer Science as a field lacks what other fields of study have: one overarching research question that clearly ties research lines together.

Different subfields in Computer Science each have their own terms, methods and practices, which are often incommensurable. It is not well defined what the foundations are that computer scientists agree on, and thus do not need further research or explanation, and what parts of the field are under investigation or discussion. In other words, Computer Science misses what

1. When discussing different subfields of computing in the remainder of this paper, I will mostly use the eight subfields proposed by Laberge, Wapman *et al.* [2022]: computational learning, (computer) systems, theory of Computer Science, Numerical and Scientific computing, Human-Computer Interaction, Inter-Disciplinary Computing, Programming Languages, and Software Engineering.

Guest describes as *[a] metaphysical commitment*: "the need to highlight what parts of theory are not under investigation, but are assumed, asserted, or essential" [Guest 2024]. The subfields that I have participated in (which do not even cover the entire breadth of Computer Science) all had greatly varying answers to these questions. So as a field, to rephrase Guest, what knowledge is Computer Science assuming without making it explicit?

In Theory[2] doing research meant writing a proof. When I was an MSc student my research work centered around a proof. When I implemented a small tool in which the proof could be explored in depth, my supervisor was ok with it, but it was clear it was not seen as a core part of the work; it was a fun extra. In Programming Languages, proofs also play a central role but implementing prototypes is more common. In Software Engineering, building tools takes a center stage, while in Computer Science Education, as in Human-Computer Interaction, people are studied using a broad variety of methods including qualitative ones, drawing on ethnography and sociology. Despite these different attitudes to knowledge creation, people in all the subfields call themselves Computer Scientists, to my increasing surprise. To my even bigger surprise, the field as a whole hardly ever discusses the question what Computer Science is, or to what issues those different approaches might lead. Ultimately observations like these, and conversations with people in other areas led me to reflect on Computer Science itself. What is *Computer Science* if different subfields hold such diverse opinions on what to study and how to study it?

Such debates, of course, happen in other fields too; how theoretical and experimental physics relate to each other is a consistent topic of inquiry in physics [Hacking 1983]. Psychology also has different research traditions, both quantitative and qualitative, that sometimes disagree on how to gather knowledge. However those fields are bound by one common question: what is the physical world made of and how does it work, or how do people think and make sense of the world around them? This brings us to the question that has been occupying my mind for the past few years: *Is there a central research programme of Computer Science, and if so, what is it?*

This question is dear to my heart, because even though I love programming, and computers and the engineering of software, I have always been interested in people, philosophy and society *also*. During my time as a Computer Science researcher, I have been told that what I was interested in was not *real* Computer Science. The remark did not pertain only to what I wanted to study (people and contexts), but also *how* I wanted to study this; with interviews, observations, ethnographic work. I realized that those methods (even though people have used them in Computer Science for decades) are not seen as Computer Science proper. "Interviewing is easy", said a senior academic to me once, "everyone can do it". If what I do is not real Computer Science, then what is?

2. When talking about Theory as in the subfield of Computer Science, I will use upper case to avoid confusion with the term *theory*.

My first attempt at answering this question for the subfield of Programming Languages was recently published [Hermans & Schlesinger 2024]. In that paper, I have examined how a masculine research culture shapes the types of research questions that are being answered, and how they are answered. The paper uses a feminist lens inspired by Carey, Jackson *et al.* [2016], and finds that research seen as mathematically difficult, and research done with quantitative methods are overvalued, at the expense of human-centered, qualitative work.

This current paper widens the scope of my research question to the entire field of Computer Science. I ask not only what questions individual researchers answer, as I did in [Hermans & Schlesinger 2024], but I am to examine how people define the field itself, its goals, aims and foundations.

I do this by analyzing four books that each aim to formulate the essence of what Computer Science is [Tedre 2014], [Primiero 2019], [Turner 2018], [Rapaport 2023]. It is important to examine what these books include, what they exclude and what parts of Computer Science they envision. This is especially important because such books not only inform researchers, but are, because of their goals of being a comprehensive overview of the field often also used as textbooks in introductory courses about the discipline.

I compare the four books by using the seminal work on the role of theory in Information Systems in which Gregor [2006] describes a similar search for the meaning of "theory". I aim to answer Gregor's questions by drawing on my own experience in the field, and examining the four books above, which attempt to draw the disciplinary boundaries and methods of computing and Computer Science.

As in my own previous work [Hermans & Schlesinger 2024], and the work of others [Ensmenger 2010], I find that the books overemphasize the mathematical and quantitative nature, and aim to place Computer Science inside of or adjacent to the natural sciences. Subfields that fit that image less, such as Human-Centric Computing, are sidelined in the books. As such, we see a second dynamic at play here that is distinct from the desire of individual researchers to do research that brings status from difficulty; the process that Computer Science is shaped not just by these individual choices, but that it needed not only to be made mathematical and scientific once, but that it needs to be continuously reinvented as such by centering the field there.

## 2 What is computer science?

In the last decade, different books have attempted to capture the essence of what Computer Science is, among them *The Science of Computing: Shaping a Discipline* [Tedre 2014], *On the Foundations of Computing* [Primiero 2019], *Computational Artifacts: Towards a Philosophy of Computer Science* [Turner

2018], and *Philosophy of Computer Science: An Introduction to the Issues and the Literature* [Rapaport 2023].

The four books each have their own view on the field. As Angius [2023] describes in their review of the most recent book (that of Rapaport):

> Interestingly, each of them analyses Computer Science from a distinct philosophical perspective, with few overlapping issues. [Angius 2023]

Tedre examines a number of debates that have taken place within the study of computing on what to study and how to study it, and uses those frictions as focal points. Tedre's analysis concerns the science of computing rather than Computer Science, though in many cases these two terms are seen as equivalent, see for example Wing [2006], who states that

> Computer Science is the study of computation—what can be computed and how to compute it. [Wing 2006]

Also, Tedre starts his inquiry from an anecdote on his own personal interests being designated as not "Computer Science". Rapaport sets as a goal to collect as many perspectives as possible, by examining both Computer Science and philosophical angles. Turner's goal on the other hand is to pinpoint the "distinctive features" of Computer Science and how it differs from other fields. Primiero finally is driven by the confusion that he observes in present day media discourse. He wants to examine how the field came to be so, allowing us to better understand what computers can bring to society. Despite their different aims and approaches, the books exhibit similarities on how they draw the disciplinary boundaries of Computer Science; they largely agree on what is being studied in the field and how it is being studied, as we will see in the remainder of this paper.

We will use the work of Gregor as a lens, who examined the role of theory in Information Systems [Gregor 2006]. Her paper asks questions about Information Systems which we use as a guide here.

## 2.1 Domain questions

Gregor asks:

> What phenomena are of interest in the discipline? What are the core problems or topics of interest? What are the boundaries? [Gregor 2006]

From the four books those of Turner and Rapaport explicitly discuss *what* Computer Science studies. Despite the fact that Computer Science is famously "not about machines, in the same way that astronomy is not about telescopes"[3]

3. A quote often attributed to Dijkstra, but first used by M. R. Fellows [1990].

*computers* are mentioned as a central concern of Computer Science. Rapaport states that Computer Science studies computers by drawing on Newell & Simon:

> Wherever there are phenomena, there can be a science to describe and explain those phenomena. There are computers. Ergo, Computer Science is the study of computers. [Newell, Perlis *et al.* 1967]

Rapaport discusses some common objections to this view, however he does not explicitly reject it himself. Rather, he comes to the conclusion that Computer Science studies many things, including computers. Rapaport compares different views on the field like in the famous fable of the blind men touching an elephant, where one decides it must be a snake on account of the trunk, while another states that it is a tree based on feeling a leg; they can't know, since they are only observing a small part of the situation. As such one of Rapaport's five central questions is "What can be computed, physically?" which surely encompasses the study of computers.

Turner also names computers as a central topic, stating "Computers in all their forms provide the concrete mechanisms that give Computer Science its physical potency" and names three topics of study that have "a strong claim to centrality": machines, theory and programming [Turner 2018].

However, even though computers are named as a core topic, in practice they are not studied within the disciplinary boundaries of Computer Science. Computer Science does not have journals or conferences "on computers". The telescope argument is used to fence off our area of interest, to exclude the messy outside world, that of people and electrical systems. If computers are studied—for example in the subfield of Computer Systems—they are studied through the abstracted lens of algorithms or proofs.

Another option for the core topics of interest is the study of *algorithms*, as stated by Rapaport, who calls the most central question is "What *can* be computed?" revealing a focus on algorithms. Turner agrees with this view and states that "abstract machines, algorithms and computational structures are [...] at the heart of the subject" [Turner 2018].

Whether or not studying *people* is currently a core area of Computer Science remains unclear in the four books, but the study of people receives just a few mentions. Tedre, interestingly enough, only mentions the study of "how people in computing really work" as a subtopic of Empirical Computer Science, and not for example, how other professionals or even citizens interact with computers. Other books don't discuss people as a possible topic of Computer Science at all.

The subfield of Human-Computer Interaction is an interdisciplinary research field focusing on interaction between computers and humans, and on the design of interfaces which improve this interaction for a wide range of different people [Kim 2015].

Even though Human-Computer Interaction's main venue, the Conference on Human Factors in Computing Systems (CHI), has been running since 1982, and is the largest and most cited of the Special Interest Groups (SIGs) [Guha, Steinhardt *et al.* 2013], Human-Computer Interaction is entirely left out of this construction of Computer Science. One can't help but tie this omission of people centered research to the fact that in Laberge's subfield classification [Laberge, Wapman *et al.* 2022] subfields studying people (Software Engineering, Interdisciplinary Computing and Human-Computer Interaction) are the three with the lowest prestige. What is cause and what is effect is hard to pinpoint, but being excluded from books on the conceptual understanding of the field of Computer Science certainly is not going to add prestige.

While other topics are discussed—Rapaport for example cites Hartmanis who states that Computer Science is the study of information, in line with the name *informatics* that is a more common name for Computer Science in Europe,[4] and Turner cites Hoare who argues that the act of programming a computer is the core of the field of Computer Science—computers and algorithms remain the most common answers to what Computer Science is about.

We can argue whether the goals of any of the books is to be descriptive or normative, most likely it is a combination of the two, however, whether this is the goal or not, such books always to a certain extent become normative, whether because of their use as textbooks in courses or read by outsiders.

## 2.2 Epistemological questions

> How can scientific knowledge be acquired? How is theory tested? What research methods can be used? What criteria are applied to judge the soundness and rigor of research methods. [Gregor 2006]

For the *Domain question* described in Subsection 2.1, we observed a field that is somewhat unified in *what* it studies. How it studies it, on the other hand, is even harder to capture. What Gregor describes for Information Systems seems to be true for Computer Science too:

> There has been little or no recognition to date [...] that the research approach adopted could vary with different types of theory. [Gregor 2006]

This is clear too in the books, two of which, Tedre and Primiero, discuss not so much what, but *how* Computer Science is studied. Tedre starts

4. The Netherlands and Italy for example use *informatica*, Germany uses *Informatik*. In general it is meant to indicate study programs with a curriculum similar to Computer Science: https://portalparts.acm.org/hippo/cecereport.pdf.

his book with a view on the history of the field, stating that computing is an ancient activity, connecting "computing science" to mathematics and calculation. From there he continues to reflect on the disciplinary identity of computing professionals, which *began to emerge around the same time as modern computing machinery*, thus connecting this identity to "computers" (even though he uses the term "Computer Science" in the book only a few times). Tedre focuses on historical debates that shaped the field, such as one on the importance of formal verification. He concludes that Computer Science consists of three different intellectual traditions, each with a different history: *the logico-mathematical, the engineering and the scientific tradition.* Indeed we see these three traditions present in Computer Science today.

Tedre acknowledges that there is Computer Science work outside of these traditions, and he names the historical, anthropological, or social science tradition as additional ones that he did not consider in his analysis, but with this choice he clearly redefines what the field means. Primiero makes a division similar to Tedre's main one, although he calls the three parts "foundations" and distinguishes *the mathematical, engineering and experimental foundation.*

Each of the different research traditions have their own ways of doing research: mathematics focuses on proofs, engineering focuses on applying knowledge to create things in the real world, while science studies the real world to create generalizable knowledge. As such, one can see how it is hard to unify these traditions into one Computer Science with a shared research philosophy.

## 2.3 Socio-political questions

How is knowledge applied?

> Is the knowledge expected to be relevant and useful? Are there social, ethical, or political issues associated with the use of the disciplinary knowledge? [Gregor 2006]

When we acknowledge the three different traditions within Computer Science and their different relationships to knowledge (are you proving it, using it, or creating it?), it is clear that how Computer Science views knowledge and the creation thereof will be hard to answer in general.

The *mathematical tradition* largely leaves application and "the real world" out of scope. The real world, of course, can be (and is) used to inform scientists on what problems to study, and findings sometimes do make it to practice, but this does not happen in a deliberate way. In general, theoretical mathematics does not see it as its task to study problems from practice, or to bring solutions to practice, and Theoretical Computer Science and Programming Languages have largely adopted this view on their responsibility.

In the *engineering tradition* however, it is much more common to work on solving real-world problems for people, and reflecting on existing solutions

as a core area of study. The engineering tradition in Computer Science is separated over different subfields. Software Engineering of course comes to mind, but researchers in Systems also aim to apply knowledge to create new operating systems, computers or algorithms. However, the practicality of solutions in these subfields are often limited to narrow metrics, adopting goals like productivity, efficiency, or error reduction. While such aspects could be studied for "the greater good", more often the result of knowledge coming from Systems or Software Engineering is applied in large corporations to produce hardware or software more cost effectively. Lastly, the *experimental or scientific tradition* studies the world and aims to create theories to understand it. The subfields of Scientific Computing and Computational Learning in some cases support other research fields with tools and algorithms and thus center applicability more than other subfields of Computer Science.

Again, the different traditions hamper a unified view on the world: are computer scientists responsible for the harm that is caused by their research? Should they take applications into account and study them, or do applications happen outside of academia, in practice, and thus not the concern of Computer Scientists? Because of the different traditions, questions like these are hard to answer for Computer Science as a whole.

## 2.4 Ontological questions

Finally we direct our attention to the core question of the philosophy of Computer Science.

> What forms do contributions to knowledge take? What is theory? How is this term understood in the discipline? [Gregor 2006]

In her excellent book "What is philosophy for?" [2018], Midgley explains that existing knowledge and theories of a field supports thinking:

> Thoughts that are now blundering around loose and detached need somehow to be drawn into a pattern—perhaps eventually into a single pattern, an all-explaining design. [Midgley 2018]

For me, Midgley here captures what is missing in Computer Science. What are the central theories of Computer Science, and how does new knowledge that computer scientists create connect to each other and to existing knowledge?

How is theory constructed in Computer Science? As with the epistemological question in Section 2.2, this is difficult to capture because it differs for the different traditions and subfields.

I will admit that for the biggest part of my research career, I did not really understand the word theory, deeper than the common sense meaning of theory such as the theory of gravity. Knowing the work of Gregor, I can see how the different traditions of Computer Science have a radically different understanding of theory (and thus of research). Gregor distinguishes four

different goals for theories: design, analysis, prediction and explanation, plus a combination of the latter two. We refer to her paper for a deeper explanation, but for purposes of readability we present a small summary.

**Theories for design** are methods to do something, for example how to design an engaging user interface.

**A theory for analysis** is a different name for an ontology or taxonomy, not meant to do anything else than to constitute a scaffold for other knowledge. An example in the field of Computer Science are Fowler's different code smells and refactorings [Fowler 1999]. They are a simple list of shapes in which code can occur and methods to change that structure without changing the meaning of the code.

**A theory for prediction** aims only to predict without an underlying model, like Moore's law predicting that number of transistors, and thus the power of integrated circuits would double every 18 months.

**Theories for explanation** finally aim to explain why something happens, such as the theory of evolution. In the natural sciences, canonical theories are mostly theories of both prediction and explanation, such as the theory of gravity.

In the *mathematical tradition*, theory is related to the theoretical work of theorems and proofs. In that tradition, most theories that are constructed are *theories for analysis* (in the meaning of Gregor), to help us make sense of the world. For example, different complexity classes like P and NP support our analysis of algorithms. While the creation of theories certainly happens, it is not common. It is more common in the mathematical tradition to prove theorem and algorithms *with respect to* existing theory, rather than to create new theories.

In the *engineering tradition*, the type of theory that is most commonly used is the *theory for design*: how to build software without errors, how to build an operating system without security vulnerabilities.

The *scientific tradition* finally uses theory in the traditional meaning, *theories for explanation and prediction.* However, Primiero (in one of the rare critical remarks of his book) states that Computer Science *simply followed the use of other sciences, but with little attention to the specific aspects of the discipline and its methodological complexity.*

## 2.5 Alternative views

Of course, alternative views of the field of computing have always been part of the field, as early as the 1970s people have examined the ways in which the scientific view shaped how students of Computer Science saw the field. For example, Weizenbaum in his book on the creation of ELIZA and his subsequent realization of the risks of chat bots, wrote:

> I am constantly confronted by students, some of whom have already rejected all ways but the scientific to come to know the

> world, and who seek only a deeper, more dogmatic indoctrination in that faith (although that word is no longer in their vocabulary). [Weizenbaum 1976]

Naur's excellent *Programming as Theory Building* [1985] describes how programming is not just a means to build a program, but also a way in which a programmer reaches an understanding of the problem under investigation. Or consider the more recent book *Software Arts* by Sack [2019] which recontextualizes concepts from Computer Science to those from the liberal arts, who also have thought of concepts such as grammar, language or logic. However, they are often placed outside of the narrative of Computer Science. I for example did not know this work before I started to write this paper, and it is systematically placed outside of Computer Science. Warren's book for example is described by MIT Press itself as "an *alternative* history of computing that places the arts at the very center of software's evolution" (emphasis mine). However, as I will argue later in the paper, works outside of Computer Science are unlikely to influence discourse on computers.

# 3 What forces shape the disciplinary boundaries of computer science?

Before I will explore the work that Computer Science currently leaves undone, I will examine reasons for the current state of the field.

## 3.1 Definition through history

One reason for the shape of the disciplinary boundaries of Computer Science as it is described in the four books examined, is that all authors look not so much at what could be done, or what should be done, but on what *has* been done. They describe what has been studied in the past, or what discussions have shaped the field. As such they only draw on topics and methods that are already in scope, and typically limit them further. We see this most strongly on Tedre, who is aware of other traditions, naming the historical, anthropological, or social science tradition, but then excludes them, rather than examining them in depth. This limit to what has been means they never attempt to imagine what Computer Science could be, or what society would need from Computer Science.[5]

5. Primiero is the only author that starts with recent headlines describing unrealistic expectations of computers, creating a desire in him to inform the general public better on what computers can't do. However, before the end of the first page, he has reduced his goal to "illustrate the basis of the mathematical and engineering tools [...] that built up towards the creation of computing machines". In other words, his stated goal is to explain to people how computers work, so that they will no longer fear them. He is not using the quoted media discourse as questions to shape the field.

In drawing on the history of the field of Computer Science, all authors overemphasize quantitative methods like mathematics and engineering, and undervalue human factors, a phenomenon well-known in Computer Science [Ensmenger 2010], [Adam 1998]. Attempts to widen the field to incorporate more human-centric methods are associated with a loss of prestige, and receive push back, from the work of Tichy in the 1990s [Tichy 1998], to our own work from a feminist lens which led to an angry person canceling their Association for Computing Machinery (ACM) membership [Hermans & Schlesinger 2024], and recent work objecting to the removal of Formal Methods from the ACM Curriculum (more on this curriculum below) [Broy, Brucker *et al.* 2024].

What seems shared between the four books is a narrow understanding of the goals defining a "philosophy" of a field. Most telling is maybe the quote by Tedre, who asks "Why have the debates not ceased over the 60 to 70 year history of the discipline" [Tedre 2014, 5]. This demonstrates a shallow understanding of what a philosophical analysis of a discipline is *for*. After all, philosophers have been debating what it means to be just, or alive, or human for thousands of years. The goal of much of philosophy is not to settle debates, but rather to open them, to place questions in a new light, or as Midgley's quote above captures, to provide new ways of thinking. Framing a debate as something to be settled quickly is exemplary of precisely the mathematical Computer Science mindset that insists that all problems can be solved. Such a point of view is preventing, rather than aiding, a solid understanding of disciplinary identity.

None of the four books aim to do more than describing different views on Computer Science in terms of topics and methods. They don't aim to define or shape the different traditions into one unified field, either in terms of topics or in terms of methods, or even reflect on what it means for a field to encompass such different ways of thinking, to what friction (or insights!) that might lead.

What also stands out is that they don't engage with traditional philosophical literature on the identity of scientific fields. For example, concepts that could have been applied, such as Hacking's view on the unity of science [1983], Foucault's archaeology of knowledge [2013], or Kuhn's paradigm shifts [1970], are not applied. Where the books mention them, it is in passing in footnotes, but not as a lens to study changes in the field. For example, Tedre attributes to Kuhn the term "paradigm" which "provides the scientific community a model of what problems and solutions look like" and the "the field's typical questions", but does not further engage with the concept, nor does he incorporate the notion of *paradigm shifts*, which according to Kuhn, always happen at once:

> the transition between competing paradigms cannot be made a step at a time, forced by logic and neutral experience. Like the gestalt switch, it must occur all at once (though not necessarily in an instant) or not at all. [Kuhn 1970, 150]

Rapaport discusses Kuhn when he explains the process of science, but he too does not apply Kuhn's concept of paradigm shifts to changes, historical or potential, inside Computer Science. Hacking is named in Turner because he coined the term "Carthesian proofs" but not for his work on definitions of scientific work. Primiero names neither. Foucault is only named in Rapaport as a generic example of a philosopher in a quoted passage.

It seems like a missed opportunity to not dive into the historical and social practices of a Computer Science comprising of so different traditions. How are these traditions in tension which each other? What ways of knowing work in one but not another? What paradigms collide when switching or combining traditions? Rather than exploring such questions, the four books aim to stratify and frame the traditions of Computer Science and reach the conclusion that it is simply and without issues, a set of three streams.

The lack of reflection on the practices and beliefs of science is understandable, since these authors come mostly from the analytical tradition of philosophy (especially Turner and Primiero). That philosophical tradition rests on the use of logical and formal methods, close to those used in mathematics and Computer Science, and from there these authors developed into computing and Computer Science. The works of Foucault, Hacking and Kuhn however is more focused on the social practices of science and scientists.

However, even if we understand the roots of the authors in different (philosophical) traditions, this, does not mean that it is without effect. Works where the disciplinary boundaries of Computer Science are discussed and contextualized are an important place to also examine the social aspects of a field, especially since knowledge on the philosophy of science is so scarcely available in Computer Science.

## 3.2 Education on research methods and philosophy of science

Another factor hampering a strong conversation on the disciplinary boundaries is the fact that it is uncommon for Computer Science students to learn about the philosophy of science, or even research methods. This is my own personal experience studying and teaching Computer Science for two decades, but also a generic view of the field.

The ACM Computer Science Curricula 2023, which aims to "inform educators and administrators on the what, why, and how to cover undergraduate Computer Science over the next decade" [Aly, Becker *et al.* 2024] defines seventeen knowledge areas, but none of these concern research methods, metascience or philosophy of science. While that might seem like a normal state of affairs for people with a Computer Science background, this is not common for other fields, not even in the natural sciences.

As a comparison, let's look at physics. The EU TUNING Qualification[6] defines nine core elements for a physics Bachelor's program, one of which is "Experimental design and scientific investigation" and another is "Scientific culture", stating a candidate should be able to:

> Describe the main traits of the historical and epistemological development of physics and relate them to changes and/or issues in technology, society, and the rules of the scientific community. [Pantano & Cornet 2018]

While according to the four books it remains an open question whether Computer Science is a science, by adopting the term *science* and with the common placement within faculties of science, computer scientists do at least to a certain extent identify Computer Science as a science. This happens even in countries in which the field is called "informatica" or "Informatik". Computer scientists certainly derive benefits from this status, such as financial benefits; in most places the natural sciences are better funded than the humanities or the social sciences. However, despite this placement and corresponding benefits, the comparison of the undergraduate programs of Computer Science and physics reveals a large gap in attitude to teaching research methods and metascience.

## 3.3 A focus on difficulty

A dynamic which shapes the field which is also clearly visible in the four book is a focus on research seen as difficult. As I have extensively discussed in a previous paper, Computer Science follows other fields of the natural sciences by emphasizing difficulty as a status symbol:

> Hard work, which is understood as mathematical work such as writing proofs, or work that pushed the boundaries of programming in ways that requires difficult tools that not everyone can use, like proof assistants. [Hermans & Schlesinger 2024, 206]

In that paper I draw upon the work of Carey, Jackson *et al.* [2016], called "Glaciers, gender and science" which examines the masculine culture of glaciology. Reading Carey's paper I was struck by similar undercurrents in the PL community. Valuing hard, technical, work *over* work done for the sake of helping others is a masculine discourse. Carey *et al.* name examples of exclusionary data gathering around glaciers: more data is collected about hard-to-climb mountains, while data from women in agriculture, whose data is not an adventure, is collected to a lesser extent, distorting our views on glaciers. In Computer Science, we see similar dynamics.

6. An initiative to help serve as "for points of reference, convergence and common understanding" for European universities, see https://www.unideusto.org/tuningeu/.

## 3.4 Theory versus theory

One tiny but important issue that is hindering our thinking on the matter is a lack of a coherent understanding of the concept of *theory*. Any work that computer scientists would want to do in broadening the goals and scopes of Computer Science will inevitably need a solid theoretical grounding. But what *is* a theory? As described above, the word theory can mean many different things to many different people, and our field lacks a clear overview as the one by Gregor. Gregor's classification helps us to understand what types of theories Computer Science produces, and aims to produce. Certainly theories for analysis are common, classifications of different algorithms, security flaws or ways of writing code. Some subfields also build theories for prediction, in Software Engineering for example. The theories that researchers in Software Engineering build are often descriptions for phenomena observed, such as "builds [that happen] on Wednesday are likely to pass certification" [Hassan & Zhang 2006] while other papers report that "The likelihood P(bug) that a change will induce a fix is highest on Friday" for two large open source projects [Sliwerski, Zimmermann *et al.* 2005]. In this community however, presenting, or hypothesizing about, a theory for explanation for those results is not common.

In some subfields of Computer Science, most notably Computer Science Education, the role of theory has been examined in depth [Nelson & Ko 2018]. Nelson & Ko observe that

> Not only do we lack theories of computing knowledge, but critical for evaluating designs, we also lack robust, validated measurements of this knowledge. [Nelson & Ko 2018]

They observe how different forms of theory are used which might be at odds with each other, Computer Science Education is interested both in theories for design (how to take into account Cognitive Load Theory while making programming lessons), as well as theories for explanation (why does using Cognitive Load Theory in design help comprehension). This dual attitude towards theory, the authors say "creates tensions and trade-offs in research, given our fixed resources." Nelson & Ko finally note how limited the creation of new theories in Computer Science Education is: theories from other fields are reused and adapted, but domain-specific theories are rare:

> we have mostly applied resources towards general theories of learning and identity, while overlooking major gaps in our domain-specific theories of the content of computing knowledge. [Nelson & Ko 2018]

This finding is confirmed by a literature survey by Malmi, Sheard *et al.* [2014] on papers from 2005 to 2011. Malmi & colleagues found that 60% of papers use theories originating from other fields such as Education and Psychology, while only 16% of papers construct new theories.

An analysis of Stol, Ralph *et al.* [2016] on papers from the subfield of Software Engineering using Grounded Theory paints a similar view. Outside of Software Engineering Grounded Theory is not so well-known so an explanation might be in place. Grounded Theory (GT) is not a theory, but a qualitative research method aimed at creating new theories based on data. It is a form of inductive research applicable when little is known about a topic. Stol, Ralph *et al.* [2016] sampled 52 papers claiming to use GT or a GT based approach, however they find that:

> Eight articles presented contributions that were clearly cohesive theories consisting of constructs and relationships, while a ninth article presented a set of hypotheses that could be considered a theory. [Stol, Ralph *et al.* 2016]

The other papers employ methods commonly used in GT, but do not form theories. Rather they remain at the level of "alternative forms instead of a set of concepts and relationship". The picture painted of these two subfields can be generalized to Computer Science in a broader context: engagement with theories is rare and haphazard.

Finally, in addition to a lack of theory building, and a lack of epistemological rigor in the use of the word *theory* (whether with Gregor's classification or in another way), there is one further issue plaguing Computer Science's discussion of theory. The word theory in Computer Science already has a strong meaning and association with the subfield Theory, which is concerned with proofs and mathematics. That means that computer scientists can't easily use the word *theory* without overloading a word that has this specific meaning already. Research within Computer Science that wants to use the term theory (e.g., in *theory building* or Grounded Theory) needs to situate itself with respect to Theory continuously.

# 4 What is the computer science undone?

As we have seen above, current works on the philosophy of Computer Science constrict Computer Science to a narrow field focusing on either proving in the mathematical tradition, building in the engineering tradition or measuring and modeling in the scientific tradition. By continually rewriting the story along these lines and failing to properly encompass "softer" subfields like Human-Computer Interaction and Computer Science Education from the analysis, Computer Science keeps reinforcing its mathematical and natural sciences focused core, thus shifting both attention and legitimacy away from subfields and researchers outside of this constructed center.

This movement causes an ongoing struggle, in which researchers in subfields seen as less "core" have to do work that is seen as less important and central, and also defend their disciplinary identity as computer scientists,

a double burden. As such, work remains undone, as it cannot be done in the current discourse comfortably. This work undone falls in three different categories: methods that remain unused, the research questions that remain unanswered, and impact that remains unhad.

## 4.1 Methods unused

Firstly, the current and continual focus on science, empiricism and mathematics explicitly includes certain research methods and often also excludes others. For example, as I have previously noted [Hermans & Schlesinger 2024], the PLDI Conference on the Design and Implementation of Programming Languages interprets design narrowly.

They publish seven guidelines in their Call of Papers that authors are encouraged to use, which clearly encourage a quantitative way of thinking. "Ways of knowing that aren't numerical are hard to fit into this list; there are no guidelines about appropriate observations, or appropriate interviews." This of course does not mean papers using qualitative methods will never be accepted, but it will be harder for both authors and reviewers to decide what is acceptable evidence, and authors might be strategically or subconsciously directed away from trying.

As such the variety of research methods is lessened, weakening, I believe, a comprehensive view of topics in Computer Science. Even when studying traditional Computer Science topics which currently would fall in scope, such as examining how bugs in software systems appear and are resolved, quantitative methods like analysis of source code, repositories, or ticketing systems are much more common than human-centered methods such as interviews, observations or long-running ethnographical studies. In some cases, research methods turn into conferences and thus communities, such as the Mining Software Repositories conference, bound by *how* they study, not what, making it hard for other research methods to penetrate into these fields. This would of course be absolutely fine if there were also conferences studying the topic of bugs, issue reports and deployments in a qualitative way, yet those do not exist as (top) conferences. When they do exist they are smaller workshops with less impact. This again drives researchers to choose these methods that are more likely to lead to high profile conferences and publications.

As I summarized in our work on the subfield of Programming Languages [Hermans & Schlesinger 2024]:

> the evidence standards of the Programming Languages community explicitly value quantitative work, making qualitative work much less likely to be published.

In Software Engineering, calls for more qualitative work have been heard for decades [Seaman 1999]. However, these calls are rarely answered, even more

so, similar calls can be heard now, with researchers still asking the field to use more, more diverse and stronger research methods:

> we suggest that future research would benefit from [...] a broader set of research methods, more strongly emphasizing reflexivity, and employing qualitative guidelines and quality criteria. [Lenberg, Feldt *et al.* 2024]

## 4.2 Questions unanswered

In addition to methods unused, there are also a great number of research questions that currently remain unanswered or only partly answered. First let us again direct our attention to the topics that currently fall *within* the disciplinary boundaries of Computer Science. Without necessarily widening the scope of *what* computer scientists study, there are a lot of research questions currently unanswered.

### Algorithms

One topic currently not studied in full is algorithms. The study of algorithms is without a doubt a core area of study of Computer Science, included in one shape or form by all four books, plus a topic that no other field studies explicitly. These are studies either in theory in Theory or Programming Languages, or in practice in Software Engineering or Systems. However, in the current research culture of Computer Science, several research questions are left out of scope. In a time where algorithms are dominating people's lives in various ways, through social media timelines and large language models, it is important to study the impact of algorithms on people's lives. Their impact on physical and mental health, on politics, on the economy and on the natural world; and on the world we as academics live in ourselves, that of teaching and research [Guest, Suarez *et al.* 2025], [Binz, Alaniz *et al.* 2025].

Some work on the impact of algorithms and their social construction is being done, most notably on AI in places that can be considered within Computer Science, such as in the ACM Conference on Fairness, Accountability, and Transparency (FAccT), or in Neurips. However, this work often fails to make an impact, as experienced even by its own authors.[7] At best, this work is not taken seriously due to the dynamic of prestige outlined above, at worst, it is seen as a reason to continue the development of such algorithms, as their fairness, accountability, and transparency are also being studied.

Of course, some of this work has been done in the past, but outside of Computer Science. For example, Lave, a social anthropologist, has studied

7. https://www.linkedin.com/posts/ebender_measuring-what-matters-construct\protect\discretionary{\char\hyphenchar\font}{}{}validity-activity-7393307326834745344-yglW/.

embodied algorithms and how people use them outside of computers [Lave 1988], Ambrosetti [2016] studied algorithms before computers. Suchman [1987], also an anthropologist, studied how people develop strategies for computer use, and Turkle [2011], a sociologist and psychologist, studied authenticity in relationships to computers and algorithms.

### Computers

Although the field of Computer Science is called *computer* science, as we have seen above, Computer Science is constructed such that it is "not done" to say that Computer Science studies computers directly [Tedre 2014, p. 88]. This presents a problem, because, if computer scientists don't study computers, who does? Imagine that one is interested (as I am) in studying "computers" in a broad sense. What research field would someone fit in who wanted to study computers but through the lens of how they are shaped by popular culture [Booth 2016]? Or study how books and movies envision computers to work, both now and in the future [Larson 2008]? Or where would research fit that studies how people form bonds with computers, giving them nicknames, talking to them, cursing at them, begging for them to not crash [Reeves & Nass 1996]. These topics now remain largely unstudied in a comprehensive manner.

### Philosophy

In addition to these core topics, there are questions on topics that are now constructed to be outside of Computer Science. What could computer scientists learn from analyzing the field through lenses of theories like paradigm shifts or the archaeology of knowledge?

Such work (like this paper also) would be hard to publish inside of Computer Science, illustrated also by the fact this paper is published in a journal on the Philosophy of Science. While other fields have journals devoted to the philosophy of the field (such as *Philosophia Mathematica*) in Computer Science these efforts only exist in smaller conferences such as the biannual HaPoC conference (hapoc.org) or in generic journals such as Philosophy and Technology.

## 4.3 This is computer science

The above methods and questions are of course being used and applied outside of Computer Science, so one might ask the why I argue that these inquiries should fall within the disciplinary boundaries of Computer Science? The existence of works cited above shows the research *is* being done in media studies, ethnography or Science and Technology Studies. Why it is simply not sufficient that others do this work?

In the opening of his book, Tedre [2014], starts with the following personal anecdote, which is all too recognizable to me:

> "That's not Computer Science," a professor told me when I abandoned the traditional Computer Science and software engineering study tracks to pursue computing topics that I thought to be more societally valuable. [...] Over the years I've heard the same reason "That's not Computer Science"—used to turn down tenure, to reject doctoral theses, and to decline funding. Eventually I became convinced that the nature of computing as a discipline is something worth studying and writing about. [Tedre 2014]

Tedre here describes briefly but precisely how Computer Science's disciplinary boundaries are used to exclude both people and their research subjects of choice. Even if the work would in the exact same way be done in other places, people who find themselves within the disciplinary boundaries of Computer Science will be confined to them in their choice of topics. In times when computers, algorithms and AI are impacting the world to such a large degree, withholding us all from the knowledge from within is problematic. Tedre with his book has done laudable work to help me and others in Computer Science understand why our interests are not seen as Computer Science, by drawing on the history of the field, this paper is an attempt to not just understand but also argue for widening of the disciplinary boundaries.

Because in a perfect world of course researchers from different disciplines would interact, meet and build on each other's work. In such a world it might be sufficient that ethnographers, sociologists, psychologists and others would learn about computers. Computer scientists would then learn about this work, and implement the findings into new systems. However we do not live in such a world. We live in a world in which inter- and multidisciplinary work is challenging for several reasons, among which a lack of opportunity for career building as there is often an absence of fitting scientific journals, causing scientists to "retreat" to their own subfield [Evans 2005]. This stratification of scientific (sub)fields tends to impact people at the fringes of fields most.

The most direct impact this has is on curricula of Computer Science. As we have seen earlier in this paper, our curricula tend to be shy on courses about research methods and the philosophy of science, let alone work outside of Computer Science entirely. As such it is extremely unlikely that students learn about Lave or Turkle. I certainly never encountered any of their ideas in my studies.

It was only when I started working on recent papers [Hermans & Schlesinger 2024, Swidan & Hermans 2023] that I encountered so much research on computers that had never crossed my path. This includes some works described earlier in this paper, such as the work of Naur & Weizenbaum, but also the work on feminism and technology, such as Wajcman [1991], Harding [2004] and Fricker [2007], and work on the limits of computers and computability Dreyfus [1988, 1992].

Many of these works I found through collaborations or conversation with researchers from other disciplines, which means that indeed, this is a form of knowledge transfer that can occur, but it does not occur *likely*. It takes energy and time, which many young faculty do not have, plus you need to be interested in these ways of knowledge to seek them out specifically. Sadly for me this interest did not come so much from intellectual curiosity as from decades of hearing my work is not "real Computer Science".

These are just the struggles of the dissemination and valuation of inter- and multidisciplinary work, which have been documented elsewhere also [Vantard, Galland *et al.* 2023], [Daniel, McConnell *et al.* 2022]. However in the field of Computer Science specifically, there is a deeper issue at play. Different fields of research do not exist on a level playing field, but have been historically constructed as organized from high prestige and theoretical fields (such as physics, astronomy, mathematics) to less prestigious and applied fields (such as sociology, psychology):

This hierarchy shapes prestige, which then shapes what research is cited *prestige shapes which scientific ideas are shared and considered worthy of attention, rather than the other way around* [Pinheiro 2023].

Since Computer Science has aligned itself with mathematics quite successfully [Ensmenger 2010], in this hierarchy it is typically seen at the prestigious end. In practice this too holds up, as Computer Science departments are typically located in faculties of natural sciences or engineering schools. This unequally divided prestige impacts not only the prestige of individual researchers and thus careers, but also on knowledge sharing. Evans [2005] found that

> prestigious authors have more influence on the content of the debate because the participants in the debate must build upon the texts of the prestigious authors to legitimate their own work. [Evans 2005]

As such, flow of knowledge from field (seen as) more prestigious to less prestigious fields is more likely than the other way around. This phenomenon explains aptly why people studying Computer Science are hardly ever exposed to works on computing or algorithms from other fields. What could we, a science, possible learn from history or philosophy?

Even within Computer Science, we see similar unequal prestige. As demonstrated by [Laberge, Wapman *et al.* 2022], who we mentioned in the above too. Laberge, Wapman *et al.* [2022] find that subfields are not seen equally, but rather that more human-centered subfields such as Software Engineering, Interdisciplinary Computing and Human-Computer Interaction have the lowest prestige, so even this work inside Computer Science might not lead to enough impact when it is seen as part of "lesser" subfields. Bibliographic analysis confirms Laberge's hypothesis. Chakraborty, Sikdar *et al.* [2014] have analyzed the *inwardness* of subfields of Computer Science

as "a measure of the degree of authoritativeness of a research field", where inwardness of an individual paper is defined as the total number a paper is cited in other papers, and with that inwardness of a subfield as the total number of citations onto the field, normalized by the total papers in the field. Using this calculation, Chakraborty finds that the five Computer Science subfields with most authoritativeness, i.e., receiving most citations, are Artificial Intelligence, Algorithms and Theory, Networking, Databases and Distributed and Parallel Computing.

Understanding these dynamics for me explains a lot of my lived experiences. In the engineering school where I studied for my undergraduate degree, there were some (although very limited) courses on society and ethics. However, these courses were given by professors from a philosophy department and not taken seriously by other professors, and thus by students. This dynamic I understand is common elsewhere too, even if the courses are in study programs, they are not seen as important. This means that many Computer Science graduates will be missing important pieces in their knowledge base, as I did early in my career. For example, Naur was known to me by his work on databases, but not this theory building work, and none of the works on technology and feminism, or criticisms on the limits of computing were covered, not even courses on ethics.

Bringing some of this work into the fold of Computer Science would, as I see it, be the most realistic way to get more students acquainted with some basics on thinking about computers wider than building or programming them. We need to integrate this knowledge into Computer Science departments, and make sure that Computer Science professors cover them in the context of "serious" Computer Science courses. A course on programming languages can include a lecture on the epistemic injustice of the English eccentricity of most languages [Swidan & Hermans 2023], a course on databases can cover the breadth of Naur's thinking including work on theory building. A course on requirements engineering or testing can cover feminist perspectives on how people of different genders use computers differently [Burnett, Fleming *et al.* 2010]. Since our future graduates are currently shaping the world of computers and algorithms, and hence, the world, it is important that all of them at least have some of this knowledge and that it is not relegated to electives from outside of the Computer Science department.

Note that I do not argue that the work should be done only by researchers with a Computer Science background. Even though we might not find this a fair state of the world, Computer Science departments and researchers are, because of the view on their place in the hierarchy, often well funded. Widening the scope of what Computer Science studies might help create opportunities for researchers with backgrounds in the humanities or social sciences, while also strengthening the field of Computer Science itself. Not all work though that is needed *can* be done by people without a background in Computer Science. Some work for example on the philosophy or meta-science of Computer Science (like this paper) leans on knowledge combined with a lived

experience in the field and experience with programming, and the culture that creates it. And some forms of thinking about computers and society leans on deep knowledge of computers. For example, work building a programming language supporting Arabic [Swidan & Hermans 2023] has made me see accessibility in programming in a different light, but it requires knowledge of how programming languages are created and how compiler courses are being taught. This thinking complements existing work on digital justice from a broader perspective. Analyzing the role that computers and algorithms play from within Computer Science will open new questions and answers enriching the perspectives of researchers from outside the field itself.

# 5 Conclusion

In this paper we have examined the disciplinary boundaries of Computer Science through the analysis of four recent books, using the earlier work on theory use in Information Systems as a lens. We find that Computer Science is currently situated as studying computers and algorithms within either a mathematical, an engineering or a scientific research method. Research that aims to study computers or algorithms in a broader sense is often excluded from the disciplinary scope of the field of Computer Science, or socially constructed to the fringes of Computer Science, exemplified by the small role that the subfield of Human-Computer Interaction takes in these books on the identity of the field (and in discussions on the field at large).

The current boundaries of the field leave numerous paths unexplored, from more human-centric methods to the study of algorithms, computers and the philosophy of the field. This work is currently done, if at all, outside of the field of Computer Science, limiting its impact in the world and on computer scientists.

What I believe the study of computers and computing needs is what media scholar van Duijn calls "topic-oriented scholarship" [van Duijn 2016]:

> it takes a topic as its starting point and then seeks for the right combination of methods and expertise across multiple disciplines for approaching it, instead of starting from the set of questions and assumptions customary in a particular discipline. Thereby, it aims at making progress not just by contesting existing findings, but also by adding new perspectives on these findings. Hopefully, these perspectives will inspire researchers from both the sciences and the humanities in their future, ideally joint, research on this topic.

A better Computer Science is both possible and necessary and more topic-oriented work would be a great start.

# Bibliography

Adam, Alison [1998], *Artificial Knowing: Gender and the Thinking Machine*, London; New York: Taylor & Francis Ltd, 1st ed., doi: 10.4324/9780203005057.

Aly, Sherif G., Becker, Brett A., *et al.* [2024], Computer Science Curricula 2023 (CS2023): Rising to the challenges of change in AI, security, and society, in: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 2*, New York: Association for Computing Machinery, ITiCSE 2024, 852–853, doi: 10.1145/3649405.3659537.

Ambrosetti, Nadia [2016], Algorithmic in the 12th century: The Carmen de Algorismo by Alexander de Villa Dei, in: *History and Philosophy of Computing*, edited by F. Gadducci & M. Tavosanis, Cham: Springer, 71–86, doi: 10.1007/978-3-319-47286-7_5.

Angius, Nicola [2023], What is (the Philosophy of) Computer Science? William J. Rapaport: Philosophy of Computer Science: An Introduction to the Issues and the Literature. Hoboken, N. J.: John Wiley, Sons, 2023, *Metascience*, 33(1), 123–126, doi: 10.1007/s11016-023-00928-8.

Binz, Marcel, Alaniz, Stephan, *et al.* [2025], How should the advancement of large language models affect the practice of science?, *Proceedings of the National Academy of Sciences*, 122(5), e2401227 121, doi: 10.1073/pnas.2401227121.

Booth, Paul [2016], Computers and culture, in: *A Companion to Popular Culture*, edited by G. Burns, Chichester: Wiley, 223–242, doi: 10.1002/9781118883341.ch13.

Broy, Manfred, Brucker, Achim D., *et al.* [2024], Does every computer scientist need to know Formal Methods?, *Formal Aspects of Computing*, 37(1), 6:1–6:17, doi: 10.1145/3670795.

Burnett, Margaret, Fleming, Scott D., *et al.* [2010], Gender differences and programming environments: across programming populations, in: *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, New York: Association for Computing Machinery, ESEM '10, 1–10, doi: 10.1145/1852786.1852824.

Carey, Mark, Jackson, M., *et al.* [2016], Glaciers, gender, and science: A feminist glaciology framework for global environmental change research, *Progress in Human Geography*, 40(6), 770–793, doi: 10.1177/0309132515623368.

Chakraborty, Tanmoy, Sikdar, Sandipan, *et al.* [2014], Citation interactions among computer science fields: A quantitative route to the rise and fall of scientific research, *Social Network Analysis and Mining*, 4(1), 187, doi: 10.1007/s13278-014-0187-3.

Daniel, Kristy L., McConnell, Myra, *et al.* [2022], Challenges facing interdisciplinary researchers: Findings from a professional development workshop, *PLOS ONE*, 17(4), e0267 234, doi: 10.1371/journal.pone.0267234.

Dreyfus, Hubert [1988], *Mind Over Machine*, New York: Free Press.

Dreyfus, Hubert L. [1992], *What Computers Still Can't Do: A Critique of Artificial Reason*, Cambridge, Mass.: MIT Press.

van Duijn, Max J. [2016], *The Lazy Mindreader: A humanities perspective on mindreading and multiple-order intentionality*, Ph.D. thesis, Leiden University, URL https://hdl.handle.net/1887/38817.

Ensmenger, Nathan [2010], Making programming masculine, in: *Gender Codes*, edited by T. J. Misa, New York: Wiley, 115–141, doi: 10.1002/9780470619926.ch6.

Evans, John H. [2005], Stratification in knowledge production: Author prestige and the influence of an American academic debate, *Poetics*, 33(2), 111–133, doi: 10.1016/j.poetic.2005.02.002.

Fellows, Michael R. [1990], Computer science and mathematics in the elementary schools, in: *Mathematicians and Education Reform 1990–1991*, Providence, 143–164, URL https://api.semanticscholar.org/CorpusID:54134550.

Foucault, Michel [2013], *Archaeology of Knowledge*, London: Routledge, 2nd ed., doi: 10.4324/9780203604168.

Fowler, M. [1999], *Refactoring: Improving the design of existing code*, Reading: Addison-Wesley Longman Publishing Co., Inc.

Fricker, Miranda [2007], *Epistemic Injustice: Power and the Ethics of Knowing*, Oxford: Oxford University Press, doi: 10.1093/acprof:oso/9780198237907.001.0001.

Gregor, Shirley [2006], The nature of theory in Information Systems, *Management Information Systems Quarterly*, 30(3), 611–642, doi: 10.2307/25148742.

Guest, Olivia [2024], What makes a good theory, and how do we make a theory good?, *Computational Brain & Behavior*, 7(4), 508–522, doi: 10.1007/s42113-023-00193-2.

Guest, Olivia, Suarez, Marcela, *et al.* [2025], Against the uncritical adoption of "AI" technologies in Academia, doi: 10.5281/zenodo.17065099.

Guha, Shion, Steinhardt, Stephanie, *et al.* [2013], Following bibliometric footprints: the ACM digital library and the evolution of computer science, in: *Proceedings of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries*, New York: ACM, JCDL '13, 139–142, doi: 10.1145/2467696.2467732.

Hacking, Ian [1983], *Representing and Intervening: Introductory Topics in the Philosophy of Natural Science*, Cambridge: Cambridge University Press, doi: 10.1017/cbo9780511814563.

Harding, Sandra G. [2004], *The Feminist Standpoint Theory Reader: Intellectual and Political Controversies*, New York: Routledge.

Hassan, Ahmed E. & Zhang, Ken [2006], Using decision trees to predict the certification result of a build, in: *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, New York, 189–198, doi: 10.1109/ASE.2006.72.

Hermans, Felienne & Schlesinger, Ari [2024], A case for feminism in programming language design, in: *Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, New York: Association for Computing Machinery, Onward! '24, 205—–222, doi: 10.1145/3689492.3689809.

Kim, Gerard Jounghyun [2015], *Human-Computer Interaction: Fundamentals and practice*, Boca Raton: CRC Press.

Kuhn, Thomas S. [1970], *The Structure of Scientific Revolutions*, Chicago: University of Chicago Press.

Laberge, Nicholas, Wapman, K. Hunter, *et al.* [2022], Subfield prestige and gender inequality among U.S. computing faculty, *Communications of the ACM*, 65(12), 46–55, doi: 10.1145/3535510.

Larson, Jerrod [2008], Limited imagination: Depictions of computers in science fiction film, *Futures*, 40(3), 293–299, doi: 10.1016/j.futures.2007.08.015.

Lave, Jean [1988], *Cognition in Practice: Mind, Mathematics and Culture in Everyday Life*, Cambridge: Cambridge University Press, doi: 10.1017/cbo9780511609268.

Lenberg, Per, Feldt, Robert, *et al.* [2024], Qualitative software engineering research: Reflections and guidelines, *Journal of Software: Evolution and Process*, 36(6), e2607, doi: 10.1002/smr.2607.

Malmi, Lauri, Sheard, Judy, *et al.* [2014], Theoretical underpinnings of computing education research: What is the evidence?, in: *Proceedings of the Tenth Annual Conference on International Computing Education Research*, New York: Association for Computing Machinery, ICER '14, 27–34, doi: 10.1145/2632320.2632358.

Midgley, Mary [2018], *What Is Philosophy for?*, London: Bloomsbury.

Naur, Peter [1985], Programming as theory building, *Microprocessing and Microprogramming*, 15(5), 253–261, doi: 10.1016/0165-6074(85)90032-8.

Nelson, Greg L. & Ko, Amy J. [2018], On use of theory in computing education research, in: *Proceedings of the 2018 ACM Conference on International Computing Education Research*, Espoo: ACM, 31–39, doi: 10.1145/3230977.3230992.

Newell, Allen, Perlis, Alan J., *et al.* [1967], Computer Science, *Science*, 157(3795), 1373–1374, doi: 10.1126/science.157.3795.1373.c.

Pantano, Ornella & Cornet, Fernando [2018], Guidelines and Reference Points for the Design and Delivery of Degree Programmes in Physics, Tuning Educational Structures in Europe, European Commission – CALOHEE project, URL https://www.calohee.eu/wp-content/uploads/2018/12/1.5-Guidelines-and-Reference-Points-for-the-Design-and-Delivery-of-Degree-Programmes-in-Physics-Reader-v3.pdf.

Pinheiro, Diogo L. [2023], The construction of academic prestige and its role in knowledge circulation, in: *Routledge Handbook of Academic Knowledge Circulation*, edited by W. Keim, L. Rodriguez Medina, R. Arvanitis, N. Bacolla, C. Basu, S. Dufoix, S. Klein, M. Nieto Olarte, B. Riedel, & C. Ruvituso, London: Routledge, 369–379, doi: 10.4324/9781003290650-35.

Primiero, Giuseppe [2019], *On the Foundations of Computing*, Oxford; New York: Oxford University Press, doi: 10.1093/oso/9780198835646.001.0001.

Rapaport, William J. [2023], *Philosophy of Computer Science: An Introduction to the Issues and the Literature*, Hoboken: John Wiley & Sons.

Reeves, Byron & Nass, Clifford [1996], *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Pla*, Chicago: The University of Chicago Pres.

Sack, Warren [2019], *The Software Arts*, Software Studies, Cambridge, Mass.: MIT Press, doi: 10.7551/mitpress/9495.001.0001.

Seaman, C.B. [1999], Qualitative methods in empirical studies of software engineering, *IEEE Transactions on Software Engineering*, 25(4), 557–572, doi: 10.1109/32.799955.

Sliwerski, Jacek, Zimmermann, Thomas, *et al.* [2005], When do changes induce fixes?, in: *Proceedings of the 2005 International Workshop on Mining Software Repositories*, New York: Association for Computing Machinery, MSR '05, 1–5, doi: 10.1145/1083142.1083147.

Stol, Klaas-Jan, Ralph, Paul, *et al.* [2016], Grounded theory in software engineering research: A critical review and guidelines, in: *Proceedings of the 38th International Conference on Software Engineering*, ACM, ICSE '16, 120–131, doi: 10.1145/2884781.2884833.

Suchman, Lucy A. [1987], *Plans and Situated Actions: The problem of human-machine communication*, New York: Cambridge University Press.

Swidan, Alaaeddin & Hermans, Felienne [2023], A framework for the localization of Programming languages, in: *Proceedings of SPLASH-E '23*, New York: ACM, SPLASH-E 2023, 13–25, doi: 10.1145/3622780.3623645.

Tedre, Matti [2014], *The Science of Computing: Shaping a Discipline*, Boca Raton: Chapman & Hall/CRC.

Tichy, W. F. [1998], Should computer scientists experiment more?, *Computer*, 31(5), 32–40, doi: 10.1109/2.675631.

Turkle, Sherry [2011], Authenticity in the age of digital companions, in: *Machine Ethics*, edited by M. Anderson & S. L. Anderson, Cambridge: Cambridge University Press, 62–76, doi: 10.1017/CBO9780511978036.008.

Turner, Raymond [2018], *Computational Artifacts: Towards a Philosophy of Computer Science*, Berlin; Heidelberg: Springer, doi: 10.1007/978-3-662-55565-1.

Vantard, Marylin, Galland, Claire, *et al.* [2023], Interdisciplinary research: Motivations and challenges for researcher careers, *Quantitative Science Studies*, 4(3), 711–727, doi: 10.1162/qss_a_00265.

Wajcman, Judy [1991], *Feminism Confronts Technology*, Cambridge: Polity.

Weizenbaum, Joseph [1976], *Computer Power and Human Reason: From Judgement to Calculation*, San Francisco: W.H.Freeman & Co Ltd.

Wing, Jeannette M. [2006], Computational thinking, *Commun. ACM*, 49(3), 33–35, doi: 10.1145/1118178.1118215.