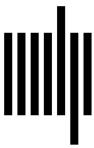
Insolvent

How to Reorient Computing for Just Sustainability

By: Christoph Becker

Citation:

Insolvent: How to Reorient Computing for Just Sustainability
By: Christoph Becker
DOI: 10.7551/mitpress/14668.001.0001
ISBN (electronic): 9780262374668
Publisher: The MIT Press
Published: 2023



The MIT Press

3

THE MYTHS OF COMPUTING

Computational thinking is a way humans solve problems.
—Wing (2006)

For me, the hardest thing to change is the cultural attitude of scientists. Scientists are some of the most dangerous people in the world because we have this illusion of objectivity; there is this illusion of meritocracy and there is this illusion of searching for objective truth.
—Gebru (quoted in C. S. Smith 2019)

In the *Communications of the ACM*, Jeanette Wing—professor of computer science (CS), director of the Data Science Institute at Columbia University and corporate VP of Microsoft Research—described the merits of what she termed *computational thinking*. Her article, widely cited in computing and beyond, argued that computer science has developed a way of thinking that is so powerful that everyone should learn it: "To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" (Wing 2006, 33). Computational thinking encapsulates computing's most profound ways of reasoning:

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. . . . Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an

appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. . . . Computational thinking is using heuristic reasoning to discover a solution. It is planning, learning, and scheduling in the presence of uncertainty. It is search, search, and more search. (Wing 2006, 33)

Wing enumerates here some of the powerful and elegant ways of reasoning that have drawn countless people into the realm of computing, including your author. Computational thinking is both about analytic abstract reasoning and about *making things*, because computer science sits at an intersection of mathematics, science, and engineering. "The constraints of the underlying computing device force computer scientists to think computationally, not just mathematically. Being free to build virtual worlds enables us to engineer systems beyond the physical world" (Wing 2006). Computational thinking thus presents a powerful toolbox of mechanisms that can be used to solve some classes of problems. And that is at the heart of how computer science understands its mission as "the foundational discipline of computing that studies the use of computers to systematically solve problems" (CS2023 2022).

The first application of such methods to real-world problems is generally attributed to the efforts of operations researchers and other scientists during World War II. Their efforts simultaneously shaped the emerging fields of operations research, game theory, artificial intelligence (AI), computer science, cybernetics, and cognitive science. A central figure in all these was Herbert Simon, whose Nobel Prize–winning work brought together some of the core ideas we will debate: rational behavior, decision-making, problem-solving, design, social planning, and the nature of human thought. By examining what Simon (1962) termed the "architecture of complexity" in naturally occurring and artificial systems, this work produced elegant approaches to managing complexity through design principles around decomposition, modularity, abstraction, and problem representation (e.g., Parnas 1972; Simon 1977). Concepts such as recursion and heuristic search have allowed computing to tackle enormously complex tasks, decompose them carefully into relatively independent subtasks, build reliable systems out of unreliable components, grow layers upon layers of abstractions to build up complex processes based on simpler ones, effectively separate concerns that could be disentangled, and merge heterogeneous insights from diverse information sources in highly modular networked systems of

algorithms with known complexity. These concepts are at the heart of all conceptions of computational thinking (Denning 2017). They also enable computing to play an important role in our understanding of complex systems such as the Earth's climate.

Computational thinking (CT) refers to "the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms" (Aho 2012, 834–835; Denning 2017). Tedre and Denning (2019) describe it as "the mental skills and practices for designing computations that get computers to do jobs for us, and explaining and interpreting the world as a complex of information processes" (4). They caution enthusiasts: "Computational thinkers need to develop enough experience and skill to know when jobs are impossible or intractable, and look for good heuristics to solve them" (8). It is instructive that the limitations they identify pertain to computational complexity, intractability, and the lack of semantics, not to any conceptual limitations of CT that may require other forms of human reasoning such as ethical judgment.

Learning to think computationally can be an empowering experience. Many computing educators believe that the core of learning computer science is learning to solve problems by developing algorithms (Peters 2019; Malazita and Resetar 2019). This "significance of learning how to write algorithms to solve problems is emphasized over the particular technical skills . . . that students learn in computer science" (Breslin 2018, 98). The core value of that skill is problem-solving: "Through discussing, learning, and practicing these particular modes of thought, students learn to understand and build within the computational universe. They learn to create algorithms, focusing on creating step by step instructions that solve particular problems" (Breslin 2018, 105).

Wing (2006) argues that CT will become so inevitable and ubiquitous that it will disappear into the background. "Computational thinking will have become ingrained in everyone's lives when words like algorithm and precondition are part of everyone's vocabulary . . . and when trees are drawn upside down" (34). In the displacement of the upright tree, the metaphor travels back to uproot our representations of nature—perhaps not such a utopian prospect. But if CT is so powerful and if, as the last chapter suggested, computing as a practice and a theory is already becoming aware of its implication in sustainability and justice, is it not enough to simply

step aside and let computing deploy its formidable powers of reasoning, innovation, development—and design—toward solving the world's problems? After all, many well-funded initiatives are doing this already under umbrellas such as "technology and society." The central organizing metaphor often found in these initiatives is technology-driven problem-solving (Pal 2017). Because problems can be represented as symbols, the real world is *rendered technical* (Breslin 2018) into problems of a *domain* of computing (Ribes et al. 2019; Ribes 2019). What could possibly go wrong?

The previous chapters showed that a lot has gone wrong so far. This and the following chapter trace historically grown concepts central to the field of computer science and explore how they inhibit the potential of those involved in systems design to think critically, reflexively, and inclusively through the situations they are facing in their design practice. By doing so, the chapters illustrate the reasoning by which even good intentions often end up reinforcing the status quo, increasing computing's debt while doing nothing to change the process by which that debt is foisted onto others. In pursuing this argument, this chapter builds on a formidable succession of critics of thinking and designing in computing (e.g., Winograd and Flores 1986; Agre 1997b; Suchman 1987; see also Bardzell 2010).

METAPHOR AND MYTH

When students of computing learn to think computationally by learning how to program, they make sense of this new symbolic domain by reference to the concepts they know. In her in-depth study of "the making of computer scientists," Breslin (2018) explains the role of programming languages in establishing the structures by which students of computer science learn to think:

> Language operates to constitute reality in particular ways, to create worlds of meaning and implication. . . . Students and professors speak of these worlds as though they exist in space, beyond the physical space in memory and computational time that a program takes up. Programs and code are talked about as though they have a shape and substance. For example, students are told functions . . . have a "territory" or a "scope." Certain functions have property, variables that they own and know about, but that other functions do not. Some data structures are in the forms of trees, with branches that can be traversed breadth-first or depth-first as different searching algorithms. . . . In learning to program, students are

thus learning to become fluent in particular languages and particular modes of thought that constitute and enable particular worlds and realities. (97)

In other words, they rely on metaphor to grasp these structures. As we get used to new metaphors, we cease to perceive them as conceptual mappings—in our minds, they eventually become *reified* and seemingly detached from the original domain that the mapping concepts were drawn from. As a result, "metaphors . . . have the power to define reality. They do this through a coherent network of entailments that highlight some features of reality and hide others" (Lakoff and Johnson 1980, 157).

By focusing on one mode of thought, others can drift out of view. Timnit Gebru, a leading critical voice on ethics and racism in machine learning (ML), emphasizes that computer scientists such as herself (and Jeanette Wing and myself) are prone to certain illusions about the nature of their work. In this chapter, I identify four of these illusions, trace them to their origins, explore their implications, and point to evidence that exposes them as flawed. I will treat these illusions as myths. In its simplest sense, the word *myth* refers to a "widely held but false belief" (*Oxford English Dictionary* 2020d). Myths are important because they are often at the heart of cultural narratives. Sets of myths form historically grown networks of stories that establish norms, values, and behaviors as part of a cultural tradition. In *The Charisma Machine*, Morgan Ames (2019) summarizes the role of mythologies and ideology in technology development: "cultural mythologies [are] foundational narratives that are ritualistically circulated within groups to reinforce collective beliefs. Mythologies have an element of enchantment to them, making certain futures appear at once magical and inevitable, straightforward and divine . . . for nearly two hundred years, mythologies have been central to the way that the United States and Europe think about technology" (18–19). It is hard to overlook the resonance of these arguments in Wing:

This kind of thinking will be part of the skill set of not only other scientists but of everyone else. Ubiquitous computing is to today as computational thinking is to tomorrow. Ubiquitous computing was yesterday's dream that became today's reality; computational thinking is tomorrow's reality. . . . Computational thinking will be a reality when it is so integral to human endeavors it disappears as an explicit philosophy. . . . We'll thus spread the joy, awe, and power of computer science, aiming to make computational thinking commonplace. (Wing 2006)

As Ames (2019) continues, "cultural mythologies . . . are aspects of what social theorists call *ideologies*: the frameworks of norms, generally taken for granted and unconsciously held, that shape our beliefs and practices and that justify differences in power between various social groups . . . ideology fades into the background: . . . ideologies are as invisible to many people as we imagine water is to a fish" (19).

Wing's argument has circulated widely and reinforced collective beliefs about the foundational narrative of computing. The emotionally charged terms she uses speak to what Ames calls the "element of enchantment." My aim here is not to deny that computational thinking has merits and value. But it is worth looking deeper into how such narratives work and what they hide. Roland Barthes (1972) writes that "myth does not deny things, on the contrary, its function is to talk about them; simply, it purifies them, it makes them innocent, it gives them a natural and eternal justification, it gives them a clarity which is not that of an explanation but that of a statement of fact" (143). Vincent Mosco (2004) concludes that "according to Barthes, myth is depoliticized speech, with political understood broadly to mean the totality of social relations in their concrete activities and in their power to make the world" (30). As a result of this purification, myths become detached from their origins and turn into "congealed common sense" or "stories that help people deal with contradictions in social life that can never be fully resolved" (28–29). Mosco (2004) provides compelling reasons to pay close attention to myth:

> If myths evacuate politics, then the critique of mythology can restore and regenerate it. If the telling and retelling of the mythic story shields cyberspace from the messiness of down-to-earth politics, then the critique of the myth, told many times over in many different ways, gives new life to the view that cyberspace is indeed a deeply political place. (31)

So myths are not just wrong beliefs that can be easily rectified. In Mosco's (2004) words, "Myths are not true or false, but living or dead" (3). In some sense, they are in fact *inoculated*: "It is common to see myths presented with what Barthes called inoculation or the admission of a little evil into the mythic universe in order to protect against a more substantial attack" (34). Inoculation thus refers to minor admissions in a narrative that "serve to protect the myth by granting that there are flaws" (Mosco 2004, 34). In examining the myths of systems design, we need to stay attuned to how they are inoculated.

THE RATIONALIST CORE OF COMPUTER SCIENCE

Computational concepts are not just projected onto the real world, they are the vocabulary and grammar used to make sense of it. In other terms, they are the cartographic tools used to make the map on which reality becomes the domain of computing (Ribes 2019; Breslin 2018). Just what constitutes a familiar domain depends on the background of the individuals and groups that use metaphorical mappings. In the case of computing professionals, their education ensures that computational concepts are at the center of conceptual frameworks and mappings. These concepts are unabashedly positivist (Easterbrook 2014b), grounded in the rationalist tradition. This is reflected in CS education (Raji, Scheuerman, and Amironesei 2021). Winograd and Flores (1986, 14) summarize this tradition by illustrating how it approaches a problem:

> The rationalistic orientation can be depicted in a series of steps:
>
> 1. Characterize the situation in terms of identifiable objects with well-defined properties.
> 2. Find general rules that apply to situations in terms of these objects and properties.
> 3. Apply the rules logically to the situation of concern, drawing conclusions about what should be done.

In this tradition, reasoning is deductive and symbolic, and the stepwise process above still characterizes the core of what computer science students are taught today:

> When we asked Computer Science faculty what CS education is "about," we were told either "algorithmic thinking" or "computational thinking." These were functionally interchangeable, and were generally categorized as a combination of:
>
> – breaking down complex problems into smaller, more tractable components
> – "seeing through" the mess of reality in order to focus on "only the details that are needed"
> – using step-by-step decision-making processes, generally by using logic gates or other formalizable decision trees, to solve a problem
> – finding an appropriate process will lead to appropriate solutions. (Malazita and Resetar 2019)

This view of problem-solving is central to the mindset and educational production of computer science subjects who render the world technical *as problems* (Breslin 2018, 98–152). Breslin (2018) reports that the students she studied "took the significance of algorithmic problem-solving to heart.

Even students in the second semester of their first year emphasized how they had learned to think algorithmically, to analyze a problem, to break down a problem to smaller steps, and to devise a solution with step-by-step instructions for a computer to follow" (99). But the concepts and rules that computational thinking supplies are insufficient in at least two ways. First, they are inadequate for developing a grasp of complex real-world situations composed of multiple interrelated factors and for identifying possible interventions (Easterbrook 2014b). And second, CT is severely limited by its inability to consider and address the social and political foundations on which computing practices operate. By treating the world as something that can be computed, its presumed ontology often denies the validity of such concepts as solidarity, freedom, purpose, determination, and will, or denigrates them into the appendix as "soft issues," treated as an afterthought. Marcuse's (1964) words ring true: "Many of the most seriously troublesome concepts are being 'eliminated' by showing that no adequate account of them in terms of operations or behavior can be given" (14–15).

Breslin's (2018) account of computer science education resonates with this characterization. Right from the start, "Students are encouraged to think critically. . . . Yet, in the end they must do so within the rules of the world and follow them, otherwise . . . they will not be able to play the game" (Breslin 2018, 97). That game is defined by the ways in which modeling, abstraction, and algorithmic reasoning renders the world technical. Abstractions, data structures and object representations often represent real world constructs. "Through these modular computational worlds, things and relationships become explicitly specified and solidified into stable representations. Moreover, these representations work as ways of developing 'solutions' to predefined 'problems'" (Breslin 2018, 109). But "the worlds that computer scientists build are both filtered reflections of constructions in the actual world and performances that constitute part of that world" (Breslin 2018, 111). Computer science education produces a very particular kind of subject trained in "modes of thought that bracket ethical or sociological content from technical concern" via abstraction, a subject that "represents and enrolls certain political, ideological, epistemic, and identity positions" (Malazita and Resetar 2019, 5). These positions have no place for social awareness and political engagement; rather, they represent a "culture of disengagement" (Cech

2014). What remains is a logically sound and coherent world of reasoning, predicated on solving problems that are rendered technical by way of these models. Its pull is strong, even for those who are just observing it:

> After starting this initiation into computer science thinking and practice it becomes remarkably hard to think around. . . . During observations I was more interested in how computer science was being taught and learned. I simultaneously felt as though I had forgotten all of my anthropological theory. It did not seem to make sense in the context of computer science . . . I was continuously asked what I was trying to find out, to answer, what was my hypothesis? My response that I was interested in how gender is involved in computer science felt unsatisfactory, insubstantial. There was no problem to solve. (Breslin 2018, 139)

THE NEED FOR A CRITICAL APPROACH

If evaluating a myth was as simple as a disagreement about facts, then the normal reasoning processes of science and engineering would be perfectly capable of sorting out the misunderstanding inherent in some myths. But the myths structure how we think and talk about the world—as water surrounds fish. For this reason, the situation is more complicated: we have to lift the hood of our conceptual engines and examine *the way we think* in computing. This is among the hardest things to do. Phil Agre has famously described his struggles to extricate his own thinking from the system of thought that he had been brought up with intellectually—the research field of artificial intelligence in the 1970s—in a much-cited piece that is worth quoting at length.

> I had absolutely no critical tools with which to defamiliarize those ideas—to see their contingency or imagine alternatives to them. Even worse, I was unable to turn to other, nontechnical fields for inspiration . . . I had incorporated the field's taste for technical formalization so thoroughly into my own cognitive style that I literally could not read the literatures of nontechnical fields at anything beyond a popular level. The problem was not exactly that I could not understand the vocabulary, but that I insisted on trying to read everything as a narration of the workings of a mechanism . . . I believe that this problem was not simply my own—that it is characteristic of AI in general (and, no doubt, other technical fields as well). (Agre 1997b, 9)

Agre struggled because he interpreted arguments from a different epistemology from within his self-grown epistemology, grounded in what we now call computational thinking. We could say that Agre's (1997b) approach of

"trying to read everything as a narration of the workings of a mechanism," coupled with a "tendency to conflate representations with the things that they represent" (8), meant that he was trying to *compute* what he was reading. We can also consider it as an instance of *operationalism*, a term initially coined in physics to indicate a way of defining concepts purely via the operations by which they could be measured. "If a concept could not be operationalized—if there was no set of procedures by which its constituent terms could be measured (or at least detected)—then that concept had no place in science. If a concept could be operationalized, then it did have a place in science, meaning that the social sciences could become 'true sciences' if only they could define their terms properly" (Crowther-Heyck 2005, 65). Over time, concepts become "synonymous with the corresponding set of operations" (Marcuse 1964, 90) so that their meaning is "restricted to the representation of particular operations and behavior" (14).

Operationalism is a useful concept because it encapsulates how prior sets of metaphors and conceptual frameworks establish the rules of reasoning and perception. Operationalism does not only mean that we "operationalize" complex concepts into more tangible elements. Because those elements populate our vocabulary and define its relationships, operationalism more profoundly means the reverse: that we register reality *through* these pre-formed elements. Meanings outside these operations simply remain invisible to an operationalist mindset (Marcuse 1964). In machine learning research, this has led to a collapse of nuanced theoretical framings of justice and fairness into an operationalist definition of fairness as parity, rendering an informed discussion of the rich concepts and their ethical operationalization impossible (Jacobs and Wallach 2021). As a result, "by abstracting away the social context in which these systems will be deployed, fair-ML researchers miss the broader context, including information necessary . . . even to understand fairness as a concept" (Selbst et al. 2019).

This inability to recognize aspects that transcend predefined operations renders an operationalist mindset unable to engage meaningfully in broader discourses. Take Agre's vivid description of his challenges in reading literature from outside computer science, such as Heidegger or Garfinkel (1967):

I found these texts impenetrable, not only because of their irreducible difficulty but also because I was still tacitly attempting to read everything as a specification for a technical mechanism. That was the only protocol of reading that I knew, and it was hard even to conceptualize the possibility of alternatives . . . it finally occurred to me to stop translating these strange disciplinary languages into technical schemata, and instead simply to learn them on their own terms. This was very difficult because my technical training had instilled in me two polar-opposite orientations to language—as precisely formalized and as impossibly vague—and a single clear mission for all discursive work—transforming vagueness into precision through formalization. (Agre 1997b, 10)[1]

Agre's memory illustrates the power of metaphor to structure our engagement with literatures beyond those we are familiar with. He recognized that he was beholden to the *false consciousness* that Marcuse (1964) describes in *One-Dimensional Man* in which "ideas, aspirations, and objectives that, by their content, transcend the established universe of discourse and action are either repelled or reduced to terms of this universe" (14). Agre's reflection on his path out of false consciousness provides crucial lessons to draw on. First, he emphasized the need for critical reflection on how disciplinary language organizes the discourse.

I began to "wake up," breaking out of a technical cognitive style that I now regard as extremely constricting. I believe that a technical field such as AI can contribute a great deal to our understanding of human existence, but only once it develops a much more flexible and reflexive relationship to its own language, and to the experience of research and life that this language organizes. (Agre 1997b, 11)

Second, he emphasized the difficulty of developing this from within: "existing language and technical practice, like any disciplinary culture, runs deeper than we are aware . . . it is difficult to become aware of the full range of assumptions underneath existing practices, from technical methods to genre conventions to metaphors" (Agre 1997b, 12–13). And third, he argued nevertheless not for a disruptive break or a clean slate to begin anew, but instead for continuous, iterative, reflective and constructive engagement with and within the existing technical discourse. He hoped that it would be possible to "develop the critical tools to understand the depths below the ordinary practices of a technical field" (14). In his view,

Critical inquiry can excavate the ground beneath contemporary methods of research, hoping thereby to awaken from the sleep of history. In short, the

negative project of diagnosis and criticism ought to be part and parcel of the positive project: developing an alternative conception of computation and an alternative practice of technology. A critical technical practice rethinks its own premises, revalues its own methods, and reconsiders its own concepts as a routine part of its daily work. It is concerned not with destruction but with reinvention. (Agre 1997a, 24)

Feminist and critical scholars in human computer interaction (HCI) have followed this path (see Bardzell 2010). As Shilton (2018) writes, "Critical technical practice, as put forward by Agre, requires questioning the metaphors, forms of representation, and discourse of an entire field" (122). This present book progresses on this path with specific attention to the challenges raised by the social and temporal distance of the outcomes of design decisions; the role of requirements in navigating the space between the social and the technical; and the potential of critical systems thinking (CST) to address this challenge.

Critical systems thinking was motivated by critical theory, which emerged as opposition to the myths of traditional theory, aiming to escape what the proponents of the Frankfurt school considered the prison of "traditional theory"—that is, uncritical scientific knowledge *turned ideology* (Horkheimer 1972; Habermas 1968; Feenberg 2014; Marcuse 1964; Jeffries 2016). Later chapters will draw from critical theorists through their influence on CST and science and technology studies. For now, we only need one more concept: reification. In *The Philosophy of Praxis*, Andrew Feenberg (2014) describes reification as "the thing-like appearance of the system of practice" (262). He explains the concept's origin in Lukács, who used reification to characterize how modern societies and their institutions had come to appear as natural and immutable things.

Bureaucratic administrations, markets, and technologies are all products of our scientific age; like science they are thought to be morally neutral tools beneficial to humanity as a whole when properly used. But in reality these institutions are social products, shaped by social forces and shaping the behavior of their users. They more nearly resemble legislation than mathematics or science. Thus their claim to universality is flawed at its basis. Like legislation, they are either good or bad, never neutral. Lukács argued that when societies become conscious of the social contingency of the rational institutions under which they live, they can then judge and change them. This implication of the theory of reification distinguishes society, including its technology, from the nature of natural science. (Feenberg 2014, viii–ix)

From this macroscopic context of stratified society, reification transitioned into social life. For the Frankfurt School, reification refers much more broadly to the unproblematic appearance of social reality as objective fact beyond doubt, critique, or valuation.

> Reification means, literally, treating human relations as relations between things. In Lukács's usage, the "thing" implied in the "re" of reification is not just an entity in general but an object suited to formal rational comprehension, prediction, and technical control . . . The problem, Lukács argues, is not with scientific reason per se, but with its application beyond the bounds of its appropriate object, nature. (Feenberg 2014, 62)[2]

This last point is echoed by critical systems thinkers. The present book too does certainly not aim to disparage the scientific method or evidence or computer science, nor claim that computer science is simply ideology without merit or that any perspective on knowledge has "equal validity." Far from it. Instead, a more critical and reflective understanding of the assumptions underpinning scientific reasoning and engineering must simply be a cornerstone *of* science and engineering in the twenty-first century. And there is more to reality than computational operationalism admits. Reification explains "how the world can appear as a collection of facts" (Feenberg 2014, 86). By configuring how we think about our reality, belief systems prestructure how we perceive it and reason about it. They define what we can recognize as facts—as Horkheimer (1972) wrote, facts are *socially preformed*. There is an important historical character to that, since both the perceived objects or facts and the organ of perception are shaped by their history (200). Because they structure facts and the questions we can ask about them, our beliefs—including rigorous, logically sound belief systems such as science—turn into ideology if there is no room for a critical questioning and reflection.

When they become reified and unquestioningly adopted in a false consciousness, science and technology turn into ideology. This ideology will serve those who already have power. The challenge that critical systems thinkers tackled head on, as we will see later, is that the logical system of science, closed and coherent as it is, is incapable of justifying its own assumptions. This is not to say that it is unjustified, simply that its justification relies on concepts and arguments that are not in themselves scientific.

The aim of *dereification* is to free the false consciousness and allow it to comprehend reality more fully. What Agre described as "defamiliarizing" himself from formalized language has profound implications. He was looking for *transcendence*, understood simply as the desire to see beyond the preformed frame of reality.[3] This is not mysticism; it is a critical understanding of reason and rationality.[4] Agre recognized that reified, socially preformed computational reasoning was not enough to make sense of what AI was trying to do, and he struggled to extricate himself from this framework of thought *so that he could appreciate it more fully*. This did not mean abandoning its ideas, mechanisms, and tools—it meant incorporating them into a broader perspective in which other forms of reasoning also had a place. This book pursues parallel aims with respect to such frameworks as requirements engineering and ICT for sustainability.

The critical distance helps us notice what Werner Ulrich (1983; 1985) calls the *sources of deception* inherent to any process of discovery (1983, 22). When someone plans, or designs, for someone else, sources of deception arise most insidiously from the structure of rational argument itself. Ulrich recognized that no matter how holistic in aim, any approach to design will be selective in its explicit *and implicit* drawing of boundaries.

Even with the best intentions, selectivity will remain unnoticed unless we look for it. Ulrich's response for a critically systemic approach to design was to develop a deeply grounded system of critical heuristics that arose from the distribution of power and agency across those involved and those affected by design. We will encounter his critical systems heuristics in chapter 5. Throughout the next chapters, this book aims to systematically identify concrete sources of deception in common theories, ideas, and metaphors of computing to illustrate how we can extricate our thinking from this socially preformed frame of engagement to more fully appreciate it and deploy it more responsibly. For now, I want to return to the topic of myths by way of Feenberg's summary:

> As technologies develop, their social background is forgotten, covered over by a kind of unconsciousness that makes it seem as though the chosen path of progress was inevitable and necessary all along. This is what gives rise to the *illusion of pure rationality*. That illusion obscures the imagination of future alternatives by granting existing technology and rationalized social arrangements an appearance of necessity they cannot legitimately claim. (Feenberg 2014, 166–167. Emphais by the author)

A MYTHOLOGY

The myths I focus on are myths of logic and reason, central tenets of methodology and practice, so I take here a more conceptual view than Mosco and Ames. The myths have been questioned and dismissed before, yet they continue to have enduring appeal and effects with far-reaching consequences, like a dangerous undercurrent hidden beneath engineering methods, research questions, design pedagogy, and industry practices. They can variously be seen as stories, illusions, or theories. I single out here four myths that structure the conceptual domain of computing and its associated engineering and design practices. If any of these seem implausible, keep in mind that myths are not *believed* the way we believe the latest findings of a scientific study, or the basic laws of physics. Instead, the question is how they shape the broader currents of systems design.

The myth of *value-neutral technology* claims that software technology is neutral—that because of their abstract form, algorithms and software systems only come in touch with human values in application. Because their effects are then merely an artifact of usage choices, politics can and should be kept out of design. Moving beyond this view allows those who design to recognize their work as political and to develop sensitivity and responsibility toward the role of values in design.

The myth of *rational decision-making* conceptualizes thinking as information processing, based on a metaphor of the human mind as a computer. On the basis that the rational choice is considered optimal, it understands human deviations from rational choice as limitation, bias, and error. Moving beyond this view allows designers to recognize how their judgments transcend the narrow framing of rational decision-making, and it allows researchers and educators to account for varied forms of human judgment in design.

The myth of *objective problems* in systems design maintains that problems have an independent existence and can be discovered using applied scientific reasoning. As a result, the central emphasis in problem formulation is on correctness or consensus. Moving beyond it allows those who design to pay attention to the ethical questions in problem formulation and enable meaningful participation of those affected by their efforts.

The preceding three myths produce a fourth: What I call the myth of *solvency* maintains that *design is problem-solving* and that any negative

effects are outweighed by the benefits. Solvency's current meaning is "the possession of assets in excess of liabilities; ability to pay one's debts." (Merriam-Webster 2020b) But is not so clear that computing can or will pay its debts to the world. Moving beyond that way of thinking enables those who design to consider the wide space of alternative approaches using different metaphors.

Together, these beliefs are at the center of computing's central paradigm—the "set of theories that explain the way a particular subject is understood at a particular time" (Cambridge Dictionaries 2011). The following sections outline each myths' merits, origins, implications, and sources of deception. These will be covered in more depth by the chapters in part II. Because the struggle for a position vis-à-vis these ideas has been a defining part of the history of applied systems thinking and of design, I will center the discussion of each idea on these fields. Their debates have had a profound influence on the current discourse in computing, and their residue can be felt in the undercurrents of systems design.

THE STORY OF VALUE-NEUTRAL TECHNOLOGY

Both in industry and across the range of academic fields of computer science, technology is still widely characterized as neutral: that is, value-free and impartial. This holds for different views of what technology is—a vague abstract whole, a concrete group of technologies such as support vector machines or relational databases, and a concrete technological artifact or product such as WhatsApp or GPT-3 (T. Brown et al. 2020). In this view, purity and neutrality are seen as a virtuous absence of values, based on the traditional ideal of science as an objective enterprise (Reiss and Sprenger 2017). In this context, Boaz Miller (2014) defines the term *value* as "anything that serves as a basis for discriminating between different states of affairs and ranking some of them higher than others with respect to how much they are desired or cared about or how the personal, social, natural, or cosmic order ought to be" (70).

The theory of value-neutral technology (VNT) maintains that only in instantiation, configuration, and application do values enter the world through choices made by people with partial interests. While these choices may result in possible unfairness, the best that engineering can do is to be independent, as it were, of these choices—neutral, and thereby innocent.

WhatsApp cofounder Brian Acton, for example, did not mince words about his stance on the moral position of technology: "There is no morality attached to technology, it's people that attach morality to technology . . . It's not up to technologists to be the ones to render judgment" (Levy 2020). This view continues to be widespread, and it is reflected in academia too: "ML systems are biased when data is biased," writes a leading researcher, claiming that bias is introduced into machine learning solely through the choice of data sets.[5] Even some philosophers of technology maintain that technologies are neutral (Pitt 2014, 90).

But this view is patently false. Software systems are never truly neutral. On the contrary, designers and their organizations embody values, their design choices embed specific values in the systems through the features and qualities they construct, and software systems in turn express and enact these values through their behaviors and affordances. As Grady Booch (2014) put it, "Every line of code represents a moral decision; every bit of data collected, analyzed, and visualized has moral implications." Only some of these implications are immediately obvious. Selecting a Boolean as the type for the *gender* variable is a choice and not a neutral one: It embeds the conservative value of binary gender stereotypes into the material artifact of code, and it *will* inevitably lead to situations in which a person who does not conform to the stereotype will experience its torque. Airport body scanners, for example, tend to flag trans people as suspicious (Costanza-Chock 2020). But the opposite—to choose not a Boolean but a more appropriate type for this variable, or to refrain from collecting data about gender—is also not a neutral choice, since it explicitly considers the value of gender-sensitive technology design and enacts it in code. *There is no neutral ground here: Every choice turns values into facts*. In both cases, the designer's choice is not fully determined by facts. Another way of putting it is that the technological facts are *underdetermined*: There is room for human judgment (Feenberg 2017) and for the need to consider values in the sense introduced earlier. VNT masks this underdetermination by diminishing the role that values play in determining the shape of technological artifacts.

Proving VNT wrong in ways that are empirically persuasive for those *in computing* has been very difficult. Miller (2020) describes how persistent the idea remains despite its apparent weaknesses. He suggests that one reason for its persistence lies in the fact that its critiques are not persuasively

legible from within a science and technology perspective.[6] This brings back the specter of operationalism illustrated by Agre's experience. For each new case that is convincingly demonstrated, a concession is made—"True, *this* system is not neutral, it's terrible," "yes *that* system is not neutral—it's badly designed," we are told, and then the writers retreat to the position that *on its own*, software technology is neutral. This move admits values not into the technology itself but into the *choices* of designers and engineers, in the form of personal bias or irrational choices as opposed to the "proper," supposedly neutral way of designing things prescribed by design methods and rational decision-making. The empirical absence of neutrality is attributed to the deviation of the instance from the rule of neutrality. VNT is inoculated by the other myths. But asking for empirical falsification of VNT misplaces the burden of proof—after all, there is no empirical evidence that technology is neutral. Instead, we need to ask how this idea can be so resistant to evidence to the contrary. To do so, we will need to take a critical approach.

VNT deceives us by suggesting that design can be neutral, with three important consequences (Miller 2020): (1) It allows designers and engineers to evade responsibility; (2) it prevents critical questions; and (3) it avoids the placement of restrictions on technologies that embody unacceptable values.[7] Statements like the quote from WhatsApp cofounder Acton are always made from positions of privilege and place an undue burden of proof on those who are already disadvantaged.[8] Not admitted into these accounts is the way in which values of dominant interests *systematically* become prioritized, privileged, and embedded in artifacts, while the values of marginal interests get systematically suppressed and left out. Even when this is not happening explicitly, the outcomes of systems design are silently shaped by the values of those involved, because most common methods in the engineering and design disciplines in computing are oblivious to the role that values play in systems design.[9] The asymmetry of vulnerability that is at play between those involved and those affected suggests that the danger of moral corruption needs to be taken seriously. VNT allows those in charge to treat each instance in which a value is demonstrably violated as a bug, an accidental "problem" to be "fixed." This book takes the position that these are not bugs; they are features of the current world of computing, embodied in the idea of Silicon Valley (Schrock 2020; Liu 2020).

As Feenberg (2017) put it: "Values are not the opposite of facts . . . Values are the facts of the future" (8). Technology shapes social reality through its functions and affordances and effects, so the values that it embodies have significant reach. They do not anymore just reside in the persons who made the choices but live on in the technological artifacts that result from these choices. Therefore, "Technology designers and constructors cannot evade moral responsibility for the consequences of their products by arguing that they are morally neutral" (Miller 2020, 19).[10] In more humorous terms, here is the second entry to the Devil's Dictionary of Computing.

> **Software engineering**, n.: the social practice that converts human values, politics and moral decisions into code, features, qualities, documentation, and other technological facts.

What remains to be persuasively shown is precisely *how values become facts* in systems design, and how we can critically handle this. We will return to that challenge in chapter 6.

THE STORY OF OBJECTIVE PROBLEMS

The story of VNT presents values as sharply distinct from facts. Facts, as has been asserted throughout much of the history of science in the twentieth century, are or at least should be "value-free." The ideal of the scientific investigator is a neutral observer, and the ideal of the outcome of a scientific investigation is an objective fact that we can treat as truth. Values enter this process either as "epistemic" (such as accuracy) or as "contextual"—"moral, personal, social, political and cultural values such as pleasure, justice and equality" (Reiss and Sprenger 2017). By interpolating across observations, perspectives, and viewpoints, the observer strips these values from observations until only the facts remain. Knowledge in this view becomes "objective" by virtue of being "value-free" after the "biasing" influence of all contextual values has been removed by way of scientific reasoning.

Whether the assumption is that facts are objectively true because they correspond to an external reality or because there is a consensus mechanism that establishes when they should be regarded as true, *facts* are regarded as starting points of technological activity. Facts are what

needs to be discovered with various scientific instruments to establish the basis for technological development.[11] The most important fact that must be established early on in design is the definition of *the problem*. The etymology of the word *problem* leads back to ancient Greek, literally "a thing put forward," then in Old French (via Latin) morphing to mean "a difficult question proposed for solution." Today, the dictionary defines it as "a question raised for inquiry, consideration, or solution" (Merriam-Webster 2020). But even though we *know* that the problem is *put forward* by someone, we are quick to handle it discursively as if it had an actual existence. Its status of objectivity establishes it as (1) correct, (2) value-neutral, and (3) independent of the observer.

This idea of objectivity manifests as the *myth* of objective problems when the problem concept itself becomes reified—when *the problem* is taken as a thing to exist in the real world and treated as an object in itself, independent of the observer. This happens more frequently than we like to admit. *"What is the problem?"* already suggests that *there is* (exactly) one problem in existence, out there, if only we could have an accurate representation *of it*. In this objectivist view of problems, "problems have an autonomous existence that does not depend on any subject's knowledge, although someone must be aware of their existence . . . in the empiricist tradition, formulating a problem is viewed as analyzing reality, not as searching for goals" (Landry 1995, 321). Conceptually, saying "the problem is X" is a shortcut that implies (a) how the world currently is, (b) what is wrong about that, and (c) how we would know that it has been fixed. When we forget that this is a shortcut, the problem concept has been reified. When we remember it, we allow ourselves to recognize the value judgments inherent in the framing; we can make transparent the politics of the situation; and we can introduce fairness into the process by which problems are articulated and selected as the basis of design work.

The history of applied systems thinking provides insightful lessons of how the understanding of the nature of *problems* has shifted in the twentieth century (M. Jackson 2003; Flood and Jackson 1991a). Two major turns, each building on the previous, represent the evolution of thinking about problems and problem-solving (Flood and Jackson 1991c). Surprisingly perhaps, these shifts have not been fully replicated within the design-oriented disciplines in computing, despite long-standing debates.

In traditional systems analysis and systems engineering (Jenkins 1969), which arose from the paradigmatic problem-solving discipline operations research, problems are real and given objects. Problem statements must correctly capture the problem, in the tradition of natural and physical sciences, and the focus is to find the most effective and efficient way to *solve the problem*. Problem specifications are the central starting point because "all problems ultimately reduce to the evaluation of the efficiency of alternative means for a designated set of objectives" (Ackoff 1958). This approach runs into massive difficulties as soon as the domain in which problems are located is social. In a given situation, different stakeholders will identify different problems based on their background, including their disciplinary training. This is why in Ackoff's example of an old woman dying on the stairs, a doctor, a social worker, a family member, and an economics professor see a medical, a social, a financial, and an economic "problem" (Ackoff 1999). None of these four problems exist independently of the observer: In the real world, a poor woman had a heart attack walking up the stairs of a home that didn't meet accessibility standards for cost reasons, and she died because of the lack of affordable medical care in the vicinity of the social housing project that the home is a part of. As soon as anyone speaks of this situation in terms of "the social problem"—or, in the case of computing, more typically "the technical problem"—they frame and shape the subsequent discourse on its basis.

Ackoff (1999a) famously proposed that a problem can be absolved, resolved, solved, or dissolved: Absolution means ignoring the problem; resolution means taking a satisficing approach to addressing it; solution means taking an optimizing approach; and dissolution of a problem means "redesigning the system that has it" (115). But in discussing "the problem itself," in speaking of "errors of conceptualization," and in attributing ownership of the problem to "the system," he reinforced the idea of objective problems. The language he used still implied a sequence of problem recognition, problem definition, problem-solving that is represented by traditional systems analysis (Checkland 1981, 155). He overlooked that problems do not have a material existence *out there*: What exists in his story is a *situation* in which an old woman has died of a heart attack on a staircase. *The problem* is a concept that the observers in the situation use to frame their understanding of *what to do*. The framing of the problem

is inevitably based on a set of perspectives, concepts, and assumptions that are bound up with the standpoint of the observer stating the frame. Ackoff conflated the model with the reality it was meant to represent. He conflated "having the problem" with "framing a problem."

In the world of puzzles and mathematical problems, where the methods of operations research came from, the problem concept sits on the same logical plane as the situation it describes. There is no fracture (Agre 1997b). When we take the problem concept from an abstract realm of solvable puzzles into the real world, however, the mapping of the concept will lead us to register certain regular features of reality while masking others. It will shape how we render the world technical, and we must not mistake the rendered map for the territory. Metaphors are a central part of the social preformation of frames we use to articulate problems, so it is worth developing systematic attention to that: "spell out the metaphor, elaborate the assumptions which flow from it, and examine their appropriateness in the present situation" (Schön 1979, 138).

The fundamental turn of soft systems thinking (SSM) was a shift away from an objectivist understanding of the systems idea. In this epistemology, systems are no longer assumed to be real but treated as discursive constructs.[12] Problems too are not objectively given from nowhere but socially constructed (M. Jackson 2003), so Checkland (1981) advocated refraining from the early use of the problem concept in favor of a *problem situation*, "a nexus of real-world events and ideas which at least one person perceives as problematic" (316).

If what matters most about a problem is that it is correctly articulated, in the sense of a *correspondence* to facts, then by extension, the application of scientific methods is the only legitimate way of reasoning. As a consequence, those affected by problem-solving efforts are not regarded as a source of facts, only a source of data. That data needs to be turned into insights through scientific interpretation by the experts. The views and voices of those affected are marginalized (Midgley 1992). If, on the other hand, what matters most is that all who are present find *consensus* on "what the problem is," or *what should be done*, then we need a firm grasp of the nature of consensus that accounts for power imbalance, coercion, and marginalization. SSM does not offer this because it lacks the social theory to recognize coercion as a factor of influence (M. Jackson 1982), and

neither does standard current practice in disciplines such as requirements engineering (Duboc, McCord, et al. 2020). The turn to *critical systems thinking* responded to the realization that neither hard nor soft systems thinking were adequate in the space of technology design. What is left out in each case is the central question of legitimacy: How can those who design justify the implications of what they are doing to those who are affected by their intervention? This question will play a central role in future chapters. For now, I conclude with another entry to our dictionary:

> **kick-off**, n.: the short period in which all active project participants succumb to the illusion that they agree on what the project purpose is.

The myth of objective problems joins VNT in supporting an abdication of responsibility over technological choices: From both angles, rational methods play the role of guaranteeing an objectively necessary outcome that is freed from the values of the expert by virtue of scientific reasoning. Those who design only need to perform the task of "objective reason" by making rational decisions.[13]

THE STORY OF RATIONAL DECISION-MAKING

Since the joint emergence of artificial intelligence and computing as disciplines, the idea of rational decision-making is founded on the premise that human decision-making and thought can be explained by reference to the way a computer processes information, because "intelligence is the work of symbol systems" (Simon 1996, 23). Concepts describing the working of a computer, which is relatively well understood and formalized because it is a designed artifact, are used to understand the elusive inner workings of the mind: "The computer is a member of an important family of artifacts called symbol systems, or more explicitly, physical symbol systems. Another important member of the family . . . is the human mind and brain. . . . Symbol systems are . . . goal-seeking, information-processing systems" (21–22).

The computer thus quickly came to provide the metaphor to describe the human mind. This core metaphor of the human mind as an information processor was established by Simon's work on cognition and

human problem-solving, which proved hugely influential in computing and the social sciences (Augier and March 2004; Erickson et al. 2013). Nobel Prize after Nobel Prize, the foundational premise of the mind as a computer anchored, shaped, and structured a wide range of complex debates about the limited degree to which humans adhere to this model (Simon 1978), the ways in which they deviate (Kahneman 2002), and the best methods to measure and improve their behavior (Thaler 2017). Wing's argument comes full circle: the human mind should now gear up its computing abilities to acquire more potent concepts of higher programming languages.

The idea of rational decisions manifests *as a myth* in systems design research, education, and practice when decision-making itself is framed exclusively by the operationalist concepts that arise from the information processing metaphor. Because the influence of the idea of rational decision-making is so pervasive, its origins are rarely cited, and its assumptions are rarely defined as succinctly and explicitly as in this software engineering paper:

> In most problems, to make a decision, a situation is assessed against a set of characteristics or attributes, also called criteria. Decision making based on various criteria is supported by multi-criteria methodologies. (Filho, Pinheiro, and Albuquerque 2016)

Indeed, multicriteria decision-making (MCDM) methods are central to computing and engineering. MCDM arose out of the mathematical theories of Bernoulli (1954) who in the eighteenth century developed principles that prescribed how a theoretical agent *should* make optimal choices between gambles under conditions of well-defined probabilistic uncertainty. This provided the foundation for utility analysis (Keeney and Raiffa 1993). By way of operations research, game theory, and early computing theory, this mathematical framework has provided a foundation for countless methods in computing fields like software engineering. MCDM methods prescribe how to analyze well-defined situations when choices have to be made to identify which choice should be considered optimal, based on the assumption that the conditions and success criteria can be specified. According to this family of theories, an agent makes a decision by evaluating a set of options against a set of weighted criteria, uses this matrix to create a ranking of options, then selects the best option out of the set.[14]

While eminently useful as a normative framework to structure the evaluation of options, the underlying theory of MCDM—now called *rationalistic* in the field of judgment and decision-making—was never validated as a descriptive framework for human thought (Tversky and Kahneman 1986). On the contrary, ample evidence demonstrates that its assumptions, predictions, and explanations are inconsistent with the reasoning processes of the human mind (Beach and Lipshitz 1993). For example, large-scale field studies in decision-making showed that high-performing professionals did not evaluate multiple options against multiple criteria to compare them, rarely ranked options, and rarely selected an option from a set. Instead, they used their highly developed perceptual skills to match cues in the environment to patterns in their experience to *generate* one plausible course of action. They then used mental simulation to predict what would happen if they pursued it, and they adapted, adopted, or dropped one action at a time in sequence until they found one that satisfied them. In doing so, they often outperformed rationalistic approaches (G. Klein 1998). In parallel, ground-breaking research in the biology of cognition concluded that "the popular metaphor of calling the brain an 'information processing device' is not only ambiguous but patently wrong" (Maturana and Varela 1992, 169; Maturana 1980). Winograd and Flores (1986) showed that this realization has far-ranging implications for computing and design and questions the value of rationalistic theories.

Despite their flaws, however, the appeal of rationalistic theories proved so strong that they underpinned most empirical research in cognitive psychology, behavioral economics, AI, and computing for decades. The behavioral assumptions of computer science are firmly grounded in this rationalistic tradition, and it is not alone in struggling to overcome it. All these disciplines owe great advances to the normative foundations laid by Simon and others in these fields but struggled for decades to overcome their inherent limitations (see chapter 7).

In systems design, the myth of rational decision-making manifests in subtle ways. For one, it provides a ready-made package of reified metaphors for how decisions supposedly happen. In truth, humans *can act* in accordance with these rational models if they choose to do so, but they also have many other forms of reasoning that such narrow, impoverished frames fail to recognize (G. Klein 1998; Gigerenzer and Selten 2001). When the conception of decision-making as choice between enumerated

alternatives according to specific objectives frames research, pedagogy, and practice, its operationalism masks anything that transcends these concepts. This leaves no room for the many expressions of human judgment and wisdom that the operationalist view fails to recognize. It is no coincidence that the field that studies human decision-making is now called *judgment and decision-making* (Keren and Wu 2015). The expanded term simply recognizes that there's more to consider than rational choice.

Yet, software engineering methods are often treated as if they were programs to be run by practitioners, even though they are more appropriately described as one of the resources that practitioners use in situated action (Dittrich 2016). The narrow framing focuses research efforts on deviations from the rational model. These are understood as "bugs" in people that can and should be "fixed," usually by reference to cognitive biases (Mohanani et al. 2018), rather than indications that the normative model may be off, wrong, or unreasonable. In chapter 7, we will see how human judgment in practice often transcends what rationalistic methods can handle and how to reorient our perspective. This is particularly important when it comes to understanding how design teams make decisions with uncertain effects at a distance, as in design decisions that affect sustainability and justice.

The alternative is already here, if we are willing to look and learn from those disciplines that have grappled with Simon's legacy. Rationalistic theories become part of a broader understanding of human reasoning, information processing is understood as distinct from human judgment, and the study of decision-making can leverage *naturalistic* approaches that examine situated action as it happens in design practice (Crandall, Klein, and Hoffman 2006; G. Klein 1998). Chapter 11 will explore how behavioral research in systems design based on this perspective can approach the challenging questions that judgment and decision-making in systems design raise when the outcomes of design practice lie at a distance. For now, a few additions to the dictionary:

> **Human**, n.: annoying reminders of the real world. See *User*.
> **Irrationality**, n.: those parts of human life to which scientific rationality and engineering have no access.
> **Judgment**, n.: that which is irrational in human reasoning.

The myth of rational decision-making supports the myths of neutral technology and objective problems. Since every rational person will supposedly arrive at the same conclusion, those who design are only executing objective reason. The myth thus dispenses with judgment, overlooks expertise, and creates the illusion that there is such a thing as a neutral perspective. Rationalizing matters this way has two simultaneous effects: First, any deviation from neutrality in design outcomes, such as bias in algorithms, can be explained away as a mistake or straying from the correct path of rational decision-making. Second, blaming the irrational human for the deviation inoculates the theory and exculpates its methods from responsibility to account for those modes of reasoning that lie outside of it.

THE STORY OF DESIGN AS PROBLEM-SOLVING

The focus on objective problems matters because, in computing, design is widely characterized as *problem-solving* (Ko 2020; Jonassen 2000), but there is no shortage of debates in design about the merit of this framing (Buchanan 1992; Dorst 2006; Holt, Radcliffe, and Schoorl 1985; Huppatz 2015). The idea of design as a search for a solution in a defined problem space shares its roots in the joint origins of computing, AI, and cognitive science in the 1950s. Herbert Simon describes problem-solving as "a search through a vast maze of possibilities, a maze that describes the environment" and adds: "Successful problem solving involves searching the maze selectively and reducing it to manageable proportions" (Simon 1996, 54). Based on the paradigmatic example of a cryptarithmetic problem, he proceeded to demonstrate that the rational solution to this problem is entirely out of reach for a human problem-solver due to restrictions on memory and processing power. This theory of problem-solving provided the foundation for ground-breaking empirical research on human problem-solving (Newell and Simon 1972). Simultaneously, it provided a lasting foundation for the design of problem-solving methods and machines in the computing disciplines (Pomerol and Adam 2006; Augier and March 2004).

Simon's conception of design as problem-solving, built on the theory of bounded rationality and the metaphor of the mind as symbol processor, also had enormous influence on the discourse of design to this day (Rosner 2018). Some design researchers long struggled against the reductive

metaphor of design as heuristic search in a defined problem space (Schön 1983; Margolin and Buchanan 1995; Cross 2006; Huppatz 2015). Cognitive studies of design activity and problem-solving demonstrated that in practice the problem space and the solution space are not given—they are constructed and co-evolve continually (Dorst and Cross 2001; Ralph 2015). "Much contemporary design research, in its pursuit of academic respectability, remains aligned to Simon's broader project, particularly in its definition of design as 'scientific' problem solving. However, the repression of judgment, intuition, experience, and social interaction in Simon's 'logic of design' has had, and continues to have, profound implications for design research and practice" (Huppatz 2015). In design-oriented HCI research, "framing current situations as problems and technological systems as solutions is common" (Baumer and Silberman 2011, 2273). A recent paper in fact characterized "95% of HCI research" activity *as* problem-solving (Oulasvirta and Hornbæk 2016). The crucial skill of choosing a problem to solve is often attributed to intuition or to inert abilities akin to the so-called geek gene.[15] As Joyojeet Pal (2017) writes, "the institutions in this endeavor extend from philanthropies and corporate social responsibility groups to academic departments and inter-government agencies unified in hope that technology can solve wicked social problems" (710).

The idea of design as problem-solving, which I call *solvency*, manifests *as myth* in systems design when problem-solving becomes the only form of engagement and problem *framing* is done uncritically or implicitly. This is highly likely when the primary organizing metaphor of the discourse is computational thinking because computational thinking often treats the problem to be solved as an unproblematic starting point provided from outside. As Breslin (2018) puts it, "what [students] learn to do with their programs or algorithms is just that, to solve problems. This approach necessitates a frame of reference as formed of problems and solutions" (105). When that frame of reference dominates, systems design becomes a narrow-minded, operationalist version of what it could be. Solvency overlooks the discursive nature of the problem concept and how it positions and frames the discourse, and it overlooks or marginalizes the existence of legitimately divergent worldviews. Its rationality redefines critical concepts that transcend the meanings of its given computational structures. When multiple problem definitions contradict each other, some are seen

as correct and others as incorrect. The crucial step of constructing, negotiating, and legitimating a problem definition to settle on is omitted.

With problems reified and taken uncritically as things, problem-solving the naturalized way of engagement, and the assumptions that the technology used and designed to solve the problem is neutral and that scientifically minded problem-solving produces objectively necessary and ideal outcomes, the stage is set for an approach to technology development that is predictably harmful but remains all too common. This approach is not necessarily limited to computing—you may argue that these myths are myths of technology, science, or enlightenment rationality. Their confluence in computing, however, is especially pronounced.

ARE THESE MYTHS REALLY STILL AROUND?

In some areas of computing or adjacent to it, on the other hand, some myths have been thoroughly discredited. I will name just a few examples among many. My concern is not with those parts of the computing discourse that have overcome misleading ideas about the nature of technology, rationality, and objectivity—but rather, with what the rest of computing can learn from this.

In design studies, Simon's framework has been long critiqued, and the process of design has been reframed (Schön 1983; Cross 2006; Dorst 1995; 2006; Dorst and Dijkhuis 1995; Margolin and Buchanan 1995; Rosner 2018). But most of this debate took place outside of computing and has not been recognized universally. In information systems and requirements engineering, soft systems methodology is often cited as an important approach for deciding what problem should be addressed by systems design in situations where many stakeholders hold diverging views (e.g., Alexander and Beus-Dukic 2009), but it has been widely misunderstood as dealing with soft systems rather than taking an interpretive epistemological position on the systems concept itself (M. Jackson 2003; Checkland 2000).[16]

Critical disciplines adjacent to computing were never directly influenced by rationalistic ideas or illusions of value-free objectivity. In contrast, they were born out of opposition to these ideas. So, when it comes to value-neutral technology, its influence in HCI is limited. This is also owed to

the long-standing conversations about values in design (Friedman 1996; Friedman and Hendry 2019; Shilton 2018; Johnson and Nissenbaum 1995; Nissenbaum 1998; 2001; Flanagan, Howe, and Nissenbaum 2008). Similarly, the politics of design have been a central topic of debate (e.g., Dourish 2010). In requirements engineering, on the other hand, the recognition of human values is a recent arrival (Thew and Sutcliffe 2017), and so is the attention to politics (Milne and Maiden 2012; Duboc, McCord, et al. 2020). In CS education, Frauenberger and Purgathofer (2019) demonstrate the values and possibilities of instilling a broader awareness of ways of thought in computing undergraduates. Other progressive voices similarly are rethinking computing education from a critical perspective (e.g., Guzdial 2020b; 2020a; Ko et al. 2020).

When it comes to rational decision-making, Lucy Suchman's work on plans and situated action (1987) is widely cited and referred to in HCI and AI, and Hutchins's work on cognition in the wild (1995) is similarly recognized (Rogers and Marshall 2017; Bødker 2006). It should be noted, however, that neither of these works directly aim to supplant the core of rational decision-making outlined earlier—instead, they address and correct some of its implications. Naturalistic decision-making research using approaches such as cognitive task analysis (Crandall, Klein, and Hoffman 2006) in HCI is commonly geared at understanding the task of *users* (Diaper and Stanton 2004), not the tasks of engineering and design practice (Zannier, Chiasson, and Maurer 2007; Becker, Walker, and McCord 2017). In software engineering, Paul Ralph (2018) has characterized the tensions between normative and empirical research and pointed to the inadequacy of traditional conceptions of design as sequential problem-solving (Ralph 2015). But as Ralph and Oates (2018) have pointed out, software engineering clings to what they call "dangerous dogmas," ideas closely related to the myths I discuss.

This suggests that these ideas remain central to the discourse and are upheld despite evidence to the contrary. As Amy Ko and her coauthors write, "many of us in the computing discipline . . . dismiss the idea that computing is anything but a value-neutral tool independent from society." In words that resonate strongly with this book, the authors call out a set of "neophilic myths: that software is always right, that software is always value-neutral, and that software can solve every problem" (Ko et al.

2020). Myths continue to remain powerful as long as they are retold, and they bring forth additional false narratives. Countering them remains important (Owens and Lenhart 2020).

CONCLUSIONS

The myths of systems design exert strong influence over what computing research and practice do by shaping how central questions are proposed, discussed, and studied. When we examine the origins and connections of myths, we can see the role they play in establishing cultural meaning. The ideas of scientific objectivity, technological neutrality, rational decision-making, and design as problem-solving have shaped the self-understanding of computing throughout the past seven decades. In systems design practice and research, they manifest *as myths* when their underlying validity is overextended into a context in which they are not empirically or theoretically supported. As myths, these ideas exert a subtle influence by shifting and distorting the frame of discussion so that crucial insights are prevented from surfacing.

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

This book was set in Stone Serif by Westchester Publishing Services.

Library of Congress Cataloging-in-Publication Data

Names: Becker, Christoph (Director of the Digital Curation Institute), author.
Title: Insolvent : how to reorient computing for just sustainability / Christoph Becker.
Description: Cambridge, Massachusetts : The MIT Press, [2023] | Includes bibliographical references and index.
Identifiers: LCCN 2022038283 (print) | LCCN 2022038284 (ebook) | ISBN 9780262545600 | ISBN 9780262374651 (epub) | ISBN 9780262374668 (pdf)
Subjects: LCSH: Electronic data processing—Social aspects. | Computer systems—Environmental aspects. | Information technology—Social aspects. | Sustainable development.
Classification: LCC QA76.9.C66 B435 2023 (print) | LCC QA76.9.C66 (ebook) | DDC 303.48/34—dc23/eng/20221121
LC record available at https://lccn.loc.gov/2022038283
LC ebook record available at https://lccn.loc.gov/2022038284