Lecture Notes in
The Philosophy of Computer Science

Matti Tedre

Department of Computer Science and Statistics, University of Joensuu, Finland

3.4   The Fundamental Question in Computing

In the previous sections three different views of computer science were discussed: computer science as a mathematical discipline, computer science as an engineering discipline, and computer science as an empirical/experimental science. Although those views of computer science differ in funda‐ mental ways, their lowest common denominator has been argued to be the question of what can be, in principle, computed with any machinery. As early as in 1969, George Forsythe argued that:

> [1969] *The question 'What can be automated?' is one of the most inspiring philosophic‐ al and practical questions of contemporary civilization.*
>
> (Forsythe, 1969)

In the January-February 1985 issue of *American Scientist*, Peter Denning defined computer science as "*the body of knowledge dealing with the design, analysis, implementation, efficiency, and applic‐ ation of processes that transform information*" (Denning, 1985). By noting that computer science does not deal with calculation only, Denning arrived at what he called the fundamental question un‐ derlying all of computer science, "*What can be automated?*". Later Denning et al. changed the question to "*What can be (efficiently) automated?*" (Denning et al., 1989).

> EFFICIENCY AND EFFECTIVENESS
>
> Although the terms *efficiency* and *effectiveness* are sometimes used synonymously, there is a certain difference between them. In purely dictionary terms, *effective* means that something can produce a desired effect whereas *efficient* has a flavor of something having a good input/output ratio. In context of algorithms, the term *ef‐ fective* usually means that the steps of an algorithm must be exact and each step must be executable in finite time (Knuth, 1997:6), whereas the term *efficient* usu‐ ally refers to the behavior of an algorithm with different sized inputs (Knuth, 1997:110; Weiss, 1997:*v*; Cormen et al., 1998:6).

The question "*What can be efficiently automated?*" is indeed one of the central questions in theoret‐ ical computer science, fundamental to empirical research on computation and computers, and cru‐ cial to the design and implementation of computing systems. However, it has been argued that since the 1980s there has been an ongoing shift of (or widening of) focus away from the machine and automation and towards uses of computers, context of computer use, activities of end users, and the group interactions of end users (cf. Grudin, 1990). That is, it has been argued that there is a clear shift from *machine-centered computing* towards *human-centered computing*. Simultaneously, the attention devoted to the social implications of computing has continued to increase.

In human-centered computing the central concern changes from probing the limits of what can be done with computing technology to asking how computing technology can meet various human needs. That change has some serious ramifications, though. If human concerns should really come first, before technical concerns, perhaps the first question to be asked in human-centered computing is no longer "what can be automated" but "what should be automated". In this section different

sides of the machine-centered view "what can be automated?" are detailed, and it is asked if the recent developments in human-centered views might alter the fundamental question.

Recommended readings concerning the question "What can be automated?" include Denning et al.'s report where the fundamental question underlying all of computing is noted (Denning et al., 1989), Kimmo Raatikainen's short essay that enlarges on Denning et al.'s questions (Raatikainen, 2004; the original essay in Finnish is Raatikainen, 1992), and Jaimes et al.'s article on human-centered views in multimedia (Jaimes et al., 2006). The COSERS report *What Can Be Automated?* (Arden, 1980) has some good texts, too. The first five chapters of Shneiderman's book *Leonardo's Laptop: Human Needs and the New Computing Technologies* offer a good rationale and introduction to human-centered computing (Shneiderman, 2002).

3.4.1   What Can Be Efficiently Automated?

In addition to his one fundamental question underlying all of computing, "*What can be automated?*", Denning listed eleven topic areas of computer science and outlined a number of fundamental questions asked in each topic area (Denning, 1985). Denning had 50 questions altogether. Yet, some of the fundamental questions in Denning's topic areas of computer science differ by nature from the single fundamental question "*What can be automated?*". In fact, in 19 out of the 50 fundamental questions that Denning mentioned, the question deals with *how* instead of *what*. Denning's 50 fundamental questions include questions such as "How can large databases be protected from inconsistencies generated by simultaneous access [...]?", "How can the fact that a system is made of components be hidden from users who do not wish to see that level of detail?", and "What basic models of intelligence are there and how do we build machines that simulate them?" (Denning, 1985). The fundamental questions seem to include about as many *how*-questions as *what*-questions.

In theoretical computer science the questions *what can be automated* and *what can be efficiently automated* fall into different topic subfields. The first question, "What can be automated?" belongs to computability theory, which studies what can be computed with different models of computation (and consequently, what can be, in principle, computed with any kind of machinery). The question "What can be efficiently automated?" belongs to computational complexity theory, which studies how much resources such as time and storage does it take to solve different kinds of computational problems. Both of those questions are asked implicitly or explicitly; and with different emphases, aims, and premises; in many other branches of computer science, too.

3.4.2   Theoretician's and Practitioner's Questions

Answers to questions that ask *what* are different from answers to questions that ask *how*. Generally speaking, natural scientists tend to ask *what* and *why*, and applied scientists and engineers tend to ask *how*. The question of what can be automated divides all possible processes into those processes that can be automated and those processes that cannot be automated. That is, every process in the world either can be automated or cannot be automated. Also the question of what can be efficiently automated divides processes into those processes that can be efficiently automated (according to some criterion of *efficient*) and those processes that cannot be efficiently automated.

Conversely, there usually are many correct answers to "*How can process* p *be automated?*". For in‐stance, process *p* could be automated using implementation *a*, implementation *b*, or implementation *c*. Implementations *a*, *b*, and *c* can all be optimal in their use of space, time, and other resources, or they can all be non-optimal in different ways. For instance, sorting of integer numbers can be done in many different ways, all of which have their advantages and disadvantages.

The question "What can be automated?" is a *theoretician's question*, and often the theoretician is not concerned about the specific technologies that might be used to automate processes, be the tech‐nologies electronics, optics, pneumatics, or magic (cf. Dijkstra, 1987). The Church-Turing Thesis is usually considered to divide processes into those that can be automated with a computer and those that cannot be automated with a computer. Although the question of computability is indeed one of the central questions in computer science, in the rest of this section the focus is predomin‐antly on what can be *efficiently* automated.

The question "How can process *p* be automated?" is a *practitioner's question*, and the practitioner needs to make a number of design choices concerning how process *p* is automated. The underrep‐resentation problem (see pages 22 and 86 of these lecture notes) is especially valid in the practition‐er's question. There is not always a single optimal way to automate process *p*. Respectively, the underrepresentation problem for a practitioner reads, "How does one choose between a number of different implementations that all automate the process *p* but that all have non-optimal aspects?".

There is no generic solution for the practitioner's version of the underrepresentation problem. If there is no optimal implementation for automating process *p*, the choice of implementation depends, among other things, on the practitioner's preferences, experience, and proficiency. The practitioner has to weigh the relative significances of non-optimal and optimal aspects of different implementa‐tions and then make a decision over the implementation. Often the implementation is not chosen right in the beginning of development, but often the implementation develops and matures during the development process.

Notions of significance, or notions of what are the most important aspects concerning a certain problem / solution, need not be just subjective "feelings", but they can be shared in the sense that all practitioners may agree about which aspects of a process are the most significant and which are less significant. Naturally, the consensus may be weaker or stronger, depending on the issue. For in‐stance, there is often a trade-off between the accuracy of the result of a computation and the time that computation takes. There may be a very strong consensus that accuracy is more important than computing time in a program that calculates the flight path of an interplanetary probe; yet there may be lesser consensus about whether accuracy or speed is the most significant aspect of a web service that finds driving routes in Europe.

Theoretical Considerations and Practical Considerations

The answer to the question "Can process *p* be efficiently automated?" is a single yes/no answer, and the number of processes in the world is essentially infinite. If one wanted to chart all processes that can be efficiently automated, it would take infinitely long to ask the same question about each and every process in the world. Therefore, theoreticians often approach the question "What can be effi‐ciently automated?" through more general questions such as "*Why can some processes be efficiently*

automated?” or “*What kinds of properties are typical of processes that can be efficiently auto –
mated?*”. Theoretician's mode of working often proceeds by proving that a single process can (or
cannot) be efficiently automated and then generalizing the proof to a class of processes that all can
(or cannot) be efficiently automated because they share some essential similarities.

For the practitioner, theoretical questions about single processes “Can process *p* be automated?” are
important: If the answer is *yes*, the practitioner asks, “How can process *p* be automated?”. Perhaps
the practitioner asks even, “Which implementation automates process *p* most efficiently?” (In the
latter question the practitioner often has to define which aspects should be optimized at the cost of
others). For instance, if the problem were the *dining philosophers problem* (see Java animation), a
practitioner might first want to know if there can, in principle, be an efficient solution for the dining
philosophers problem at all. If there can be an efficient solution, the practitioner proceeds to find
such solution and to construct a working implementation of the solution. But the practitioner, too,
can ask general questions such as “*What kinds of implementations are generally best for problems
like f(x)?*” or conversely “*What kinds of problems do certain implementations automate best?*”. For
instance, the practitioner might ask what kinds of implementations are generally best for image
compression.

TRACTABILITY AND INTRACTABILITY

Those problems that can be solved with computers in principle, can be further di –
vided into *tractable* and *intractable* problems. Problems that can be solved within
some reasonable time limits and with some reasonable memory use, are called
tractable problems. Problems that can be solved with computers in principle, but
not quite in practice, are called *intractable problems*. There is no clear consensus
of what “reasonable limits” and “in practice” mean, but it is often considered that
those problems for which the time or memory consumption grow exponentially as
the input size increases are intractable. Then again, many such problems have
been, in practice, solved for very large input sizes—for instance, the traveling
salesman's problem has been solved for the 24 978 cities in Sweden.

The emphasis on efficiency is one of the central issues for the practitioner (Denning et al., 1989).
Firstly, the practitioner faces *theoretical* considerations of efficiency, that is, tractability. If the
problem is considered to be intractable, the practitioner needs to employ very different implementa –
tion strategies than if the problem is considered to be tractable. Often intractable problems are ap –
proached with probabilistic, heuristic, or approximation algorithms that quickly yield solutions that
are very probably very good. At other times problems whose general cases are intractable can be
made tractable by setting some additional constraints for the problem.

For instance, consider the following problem: “You have a large number of items that all have dif –
ferent weight and that all are of different value. Your knapsack can hold *x* kilograms. Which items
should you choose to get the highest possible total value for items in your knapsack?” (often re –
ferred to as the knapsack problem). See Figure 5 for an instance of the knapsack problem with a
knapsack that can hold 10 kilograms, and with eight items that weigh between 1 and 8 kilograms
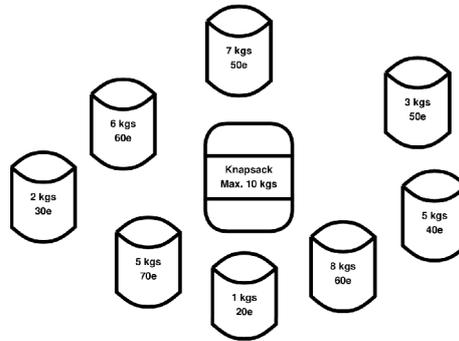and cost between 20 and 70 €.

Figure 5: An Instance of the Knapsack Problem

In general case, the knapsack problem is an NP-complete problem—that is, if one wants to find the perfect combination of items in the knapsack, there is no tractable (polynomial-time) algorithm for choosing the items. It is up to the practitioner how he or she wishes to approach that problem; for instance, heuristic strategies might be used to quickly find good combinations, and dynamic programming can, in some special cases, be used to find a perfect combination efficiently (in polynomial time and memory use). Different approaches have their limitations, advantages, and disadvantages.

Secondly, the practitioner has to accommodate to *practical* considerations of efficiency. Many proponents of the engineering-view of computer science (see, e.g., Loui, 1995; Wegner, 1976; Brooks, 1996; Hartmanis, 1993) have noted that the practicing computer scientist needs to take cognizance of real-world concerns such as operationality, usability, and maintainability of implementations. The practicing computer scientist often needs to take into account time frames, budgets, project management, material and human resources, cost-effectiveness, and resource consumption. One of the central questions for the practicing computer scientist is thereby the question of efficiency; "*How can process* p *be automated most efficiently?*". It has been has noted that the question of reliability—"*How can process* p *be automated reliably?*"—is as important as the question of efficiency (cf. Raatikainen, 2004).

Theoreticians in the field of computing are generally interested in finding classes of processes that can be efficiently automated; one of the ultimate goals of a theoretician might be to find the quintessential property that would divide all processes into those processes that can be efficiently automated and those processes that cannot be efficiently automated. One of the theoretician's tasks is to provide proofs that classes of processes either can or cannot be efficiently automated. When a theoretician devises a proof about a class of processes, the theoretician answers the question "*Why can this class of processes be efficiently automated?*".

The questions discussed in this subsection were concerned with *what* can be efficiently automated, *why* can some things be efficiently automated, *how* can one automate those things, and how to achieve *efficient* and *reliable* implementations. Many computer scientists indeed work in areas where those are the central questions. Although those questions may crystallize the computational core of computer science, their status in the definition of computer science is still ambiguous. It seems that asking any or some those questions is not a *sufficient condition* of being computer science. It might be hard to consider research associated with things such as mechanical food processors, nail guns, and watermills to be computer science even though all those things automate tasks. (Though it must be noted that Denning et al. bound computer science to automation of al-

gorithmic processes.) It is uncertain if those questions are *necessary conditions* of being computer science, either. The following subsection discusses how recent characterizations and definitions of computer science have increasingly—and some might say unfortunately—extended towards human, social, and ethical issues.

3.4.3 Human-Centered Computing

In the previous chapters of these lecture notes, the Denning Report was discussed to some extent. In their 1989 report *Computing as a Discipline*, Denning's task force noted that in the discipline of computing, science and engineering cannot be separated because of the fundamental emphasis on *efficiency* (Denning et al., 1989). The Denning Report has been widely cited and it can be con‐ sidered to be widely accepted among computer scientists. Denning et al.'s brand of computer sci‐ ence focuses on the theoretical and technical limits of machine computability, and its fundamental question asks what machines can do. That view could be called *machine-centered computing*.

Ed Lee, in the *34th IEEE Computer Society International Conference*, San Francisco, presented a definition of computer science that was much different from Denning's definition, even though Lee's and Denning et al.'s definitions were both presented in 1989. Lee wrote:

> [1989] *Computer science is the field of* human *endeavor that includes the study, the design and the use of machine based data processing and control systems to enhance* peoples' *ability to study information, perform work or explore reality.*
>
> (Lee, 1989; emphases added)

In Lee's opinion, the focus of human-made computer science is the human. He wrote, "*Neither a computer nor the teaching of computer science has any value or meaning outside of its impact on people*" (Lee, 1989). Lee's definition of computer science places the human, not the computer, at the core of computer science. In other words, computer science has no intrinsic value, but it only acquires value through its effects on people or society. This view could be called *human-centered computing*. Although it could be argued that human-centered computing was born in the 1980s when the focus of interface design began to turn towards end users (see Grudin, 1990) it can also be argued that the shift from machine-centered computing to human-centered computing is still largely uncertain and incomplete. It might be dubious to even call it a *shift*—perhaps *extension* is a better term since human-centered branches of computing are dependent on machine-centered branches of computing.

Competing Terms

There are a variety of concepts that are similar to *human-centered computing*; for instance, *new computing* (Shneiderman, 2002), *end-user computing* (Mahmood, 2002), and *user-centered techno‐ logy* (Johnson, 1998). All those terms have their strengths but also have their weaknesses: Ben Shneiderman's term *new computing* emphasizes a shift from *old computing* (which is about what computers could do) to *new computing* (which is about what users can do), but although the content of Shneiderman's new computing is rich and human-centered, the term *new computing* itself is vague. (What is new computing anyway? Today's new computing is tomorrow's old computing.) The terms *end-user computing* and *user-centered technology* both focus on the human aspects of

computing, but *user* in both terms implies that people are active in their interaction with computing technologies.

Instead of being consciously active users, people are nowadays often *not active* in their interaction with computing technologies (Kamppuri et al., 2006). Note that a large technological shift happened after Denning's report in 1989, in the period between the early 1990s and mid-2000s: In‐creased affordability, miniaturization, integration, and interoperability of information and commu‐nication technology took computing machinery from the desktop to the pocket, from cable-bound to wireless, from rare to ubiquitous, and from shared to private (Kamppuri et al., 2006b). In addition, during the period between the early 1990s and mid-2000s the amount of technology increased, its forms diversified, and information and communication technology gradually became an integral and commonplace part of many people's lives in industrial countries. Computing technologies have per‐vaded everyday life; people use computing technology when they turn off their alarm in the morn‐ing, heat their breakfast, drive their cars to work, collaborate with their colleagues, call their friends, and entertain themselves (Kamppuri et al., 2006). But computing technologies are often active also when people do not even know they are there: switching lights on and off, regulating room temper‐atures, monitoring people's moves and traffic, and so forth (Tedre, 2006b).

Although the ubiquity and the number of technologies have increased, the need for people to be *act‐ive users* of computing technology has not increased to the same extent, because new computing technologies often work without people having to be aware of them (*ubiquitous computing*). Very often people are in contact with computing technology without knowing it, and the verb *use* seems to connote some kind of conscious intention to use technology—therefore, the term *human-centered computing* might be preferred to the terms *user-centered computing* and *end-user comput‐ing* (Tedre, 2006b). (I agree that the term is debatable; see, e.g., Norman, 2005.)

3.4.4   The Ethical Question

Technological development has always entailed questions of whether some things should be auto‐mated or not. Carl Mitcham noted that skepticism about technologies is hinted already at the an‐cient myths of Prometheus, Hephaestus, and Daedalus and Icarus (Mitcham, 1994:277). Mitcham wrote that the attitudes towards technology have ranged from Ancient skepticism to Enlightenment optimism to Romantic uneasiness (Mitcham, 1994:277-297). Many technologists set aside ethical concerns from technological development by arguing that science, technologies, or technological development are neutral or value-free.

Neither the theoretician's question of *what* can be efficiently automated; nor the practitioner's ques‐tion of *how* can processes be automated reliably and efficiently; include, explicitly or implicitly, any questions about *why* certain processes should be automated at all or *is it desirable* to automate cer‐tain things. If the central concern is the machine-centered "*What machines can do?*", it might be possible to set aside ethical issues by arguing that machines have no intentions and morals, that technologies are value-free, or that theories are neither good nor evil. Some computer scientists in‐deed would like to keep ethical issues away from computer science.

However, a shift from machine-centered computing towards human-centered computing forces hu‐man-centered questions to focus. If the central concern is "*What users can do?*", the whole gamut

of ethical and social questions cannot be brushed aside. Those questions include questions such as "What should be automated?", "Should process *p* be automated or not?", "Why should process *p* be automated?", "When should process *p* be automated and when not?", "What individual or societal consequences does automating process *p* have?", "Who should decide what will be automated and what will not?", and perhaps even meta-questions such as "How can we know what should be auto‐mated?".

Shneiderman argued that the key question of human-centered "new" computing is "not whether broadband wireless networks will be ubiquitous, but how your life will change as a result of them" (Shneiderman, 2002:14). Changing people's lives is certainly not a mere technical matter but it en‐tails ethical questions of what is *desirable*. The questions of machine-centered computing are *de‐scriptive questions*, questions about "what is" or "what can be", but human-centered computing en‐tails *normative questions*, questions about "what ought to be". Those two kinds of questions are from different realms, and prescriptions of "what ought to be" cannot be derived from descriptions of "what is" (this distinction—Hume's Guillotine—is discussed later in these lecture notes).

A Technical-Theoretical-Ethical Question

In contrast to Denning's fundamental question underlying all of computing, "What can be efficiently automated?", the extended or "softened" version of the fundamental question underlying all of hu‐man-centered computing might be, *"How can one efficiently and reliably automate processes that can be automated and that should be automated?"*. This question includes the theoretician's ques‐tion, "What can be efficiently automated?", the practitioner's question, "How can things be auto‐mated efficiently and reliably?", and the human-centered question "What should be automated?". All three aspects are necessary for human-centered computing, but answers to them lie in different domains of knowledge. The first one belongs to the domain of theoretically oriented computer sci‐ence, the second one belongs also to the domains of engineering-oriented and empirical computer science, and the third one belongs perhaps to the domains of social sciences and applied philosophy (see Figure 6).
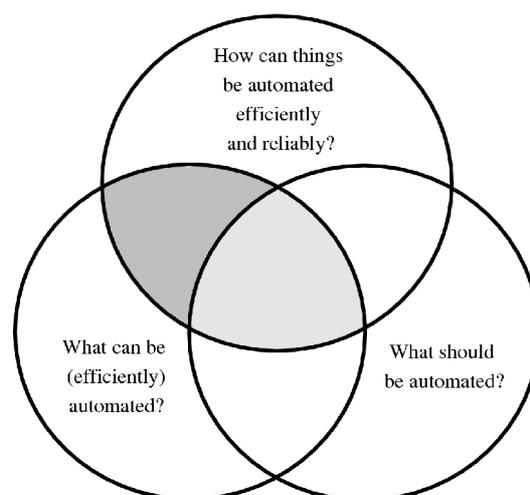

Figure 6: Three Fundamental Questions in Human-Centered Computing

The theoretician's question, the practitioner's question, and the human-centered question are por‐trayed in Figure 6. All three questions are important as they stand, and research in non-intersecting

areas is no less important than research in the intersecting areas. In machine-centered computing the production of efficient computing machinery takes place in the whole grayed intersection, but in human-centered computing responsible production of useful computing machinery takes place in the intersection marked with light gray.

However, in order to human-centered computing to really respond to human needs, it is not enough to consider only the theoretical, technical, and ethical questions. Human-centered computing also requires an understanding of the needs, wants, hopes, expectations, and wishes that people have about technology. And it also requires an understanding of the fears, threats, and anxieties that people have about technology, as well as other challenges and barriers to technology. Naturally, humanities and social sciences play an important role in human-centered computing. This charac‐ teristic of human-centered computing may seem quite questionable to theoretically and technically oriented computer scientists. They can justly ask if human-centered computing is or ought to be a part of computer science at all.

But *if* one were to agree that the field of computing is expanding towards human-centered comput‐ ing as it has been described, and *if* the theoretical-methodological-conceptual toolbox of computer science turns to be inadequate in human-centered computing, then tools need to be borrowed from other disciplines. In Denning et al.'s machine-centered computing there are three separate but irre‐ vocably intertwined branches: *theory*, *modeling*, and *design* (Denning et al., 1989). In human-centered computing there is one more branch, call it, for instance, *social studies of computing* (the term is from Kling, 1980). Social studies of computing are, to some extent, acknowledged in the ACM computing classification system (Class K: *computing milieux*).

Whose Specialty Is Human-Centered Computing Anyway?

In 1959 two influential commentators, C. Wright Mills[21] and C. P. Snow, noted a significant lack of communication and understanding between people from the humanities and social sciences, and people from science and technology (Mills, 1959; Snow, 1964[orig.1959]). Mills argued that most people who *use* technological gadgets do not understand technology, and that those people who un‐ derstand technology do not understand much else (Mills, 1959:175). Snow noted the same prob‐ lem, but the other way round—he argued that no more than one in ten highly-educated humanities people are able to discuss, for example, mass and acceleration, which are the scientific equivalents of asking "Can you read?" (Snow, 1964:15). Furthermore, Mills argued that the scientists' work is centered around "Science Machines", which are operated by technicians, and controlled by econom‐ ic and military people. (Mills coined the term *power elite* to refer to the political, military, and eco‐ nomic élite that exert a strong control over wealth and power; see Mills, 1956.)

Later Snow wrote that people with a variety of abilities, not only technologically oriented abilities, must study, control, and humanize the effects of the computer revolution (Snow, 1966). He noted that unlike our ancestors, who could not foresee the effects of the first industrial revolution, there is no excuse for the people of today to not control the computer revolution. However, it is harder to say *how* the computer revolution should be controlled or *who* should control it than it is to argue that the computer revolution should be controlled. Probably few computer scientists today would

---

21 The appendix to Mills' *The Sociological Imagination* (Mills, 1959), titled *On Intellectual Craftsmanship* is available at http://www.angelfire.com/or3/tss/millsoic.html. It is definitely worth reading.

disagree with Snow, but the question for computer scientists might rather be whether ethical and so‐ciocultural issues belong to computer science or to some other discipline.

One might argue that contemplating on *ethical* issues concerning technology is not the task of com‐puter scientists but of philosophers, sociologists, or other specialists of human and societal issues. The task of the computer scientist can defensibly be argued to be studying computations, com‐puters, or other computing-centered topics and not sociocultural topics (e.g., Dijkstra, 1987). After all, moral philosophers, sociologists, and anthropologists are experts in ethical and sociocultural is‐sues; computer scientists (in the traditional sense) are experts in theoretical and practical aspects concerning automatic computation.

A counterargument to the paragraph above is that because computer science is not detached from society, computer scientists cannot elide societal and ethical responsibility. There is a two-way re‐lationship between computer science and society: Firstly, the products of computer science are used in society, and secondly, the scientific activities in computer science are made possible by society. Receiving funding from external sources entails ethical questions, such as how to maintain autonomy of research. Effects on society and individuals entail ethical questions such as who is af‐fected by technology, how they are affected, and are such effects desirable. Ethical and profession‐al issues are nowadays acknowledged as computer science; see, for instance, the ACM Computing Classification System or Computing Curriculum 2001.

Although the status of philosophical, sociological, psychological, anthropological, and all other kinds of scholarship as a part of computer science can be disputed in machine-centered computing, many of them clearly belong to human-centered computing. What is *not* clear in human-centered computing is whether those issue belong to *computer scientists* or someone else. Though it may not be the computer scientists' task to study things like social actions, cultures, the human behavior, or ethics, computer scientists do have expert knowledge about the possibilities and limitations of com‐puting technology, and that expertise must be utilized in social studies of computing.

Clearly, human-centered computing does not take away the need for technological experts—tradi‐tional computer scientists play a central role in any decision about technological development. But in human-centered computing the questions are not only about machines; the questions are also about people. The chances are that the methodological, conceptual, and theoretical frameworks of computer science, as described in, for instance, the Denning Report, turn out to be insufficient to deal with the issues of human-centered computing. That is, the framework of computer science may turn out to be insufficient for selecting, recording, understanding, explaining, analyzing, or pre‐dicting phenomena in the field of human affairs. Co-operation between experts from different dis‐ciplines is necessary.

The debate about the inclusion of sociocultural issues under the term *computer science* has some‐times been harsh. The status and position of social studies of computing seems to be a charged, emotional issue. When the debate is brought to a head, those who argue that social issues do not belong to computer science are easily labeled as irresponsible technophiles, and those who argue that social issues belong to computer science are easily labeled as idealistic do-gooders.

It indeed seems unreasonable to expect computer scientists to become experts in sociocultural, eth – ical, economic, or other issues outside computer science. Mastering computer science is hard as it turns out. It is equally unreasonable to expect that people from other disciplines become experts in computer science. Experts, specialists, and professionals have their areas of expertise and they should do what they know best. Instead of broadly trained bricoleurs, human-centered computer science might require a working multidisciplinary combination of experts from different fields (see page 39 of these lecture notes for the terms *multidisciplinarity* and *interdisciplinarity*).

Given the significant impact of computers on all areas of life in Western societies, human-centered development trends are well justified. However, many aspects of human-centered computing are still wide open. Although the status of human-centered computing has strengthened ever since the advent of personal computing in the late 1970s, there is still today no established, shared idea of what *human-centered computing* is (see Jaimes et al., 2006 for a list of views). It is not certain at all whether human-centered computing research should belong to academic computer science in the first place. But philosophers, sociologists, psychologists, anthropologists, and all other kinds of scholars are equally reluctant to adopt parts of human-centered computing under their discipline. Perhaps human-centered computing needs neither to spawn a new interdisciplinary science nor to be included under any established field, but perhaps it should form an eclectic multidisciplinary um – brella. That is to say, maybe human-centered computing should remain a systematizing, categoriz – ing, and collecting concept instead of gaining a status as an acknowledged branch of computer sci – ence.