n the pioneering days of computer programming, computers were remote and monolithic, the programmers were the users, and the interface was provided by operators, who transformed instructions on punched cards into the programs themselves. In effect, code was being written by mathematicians, electrical engineers, and physicists who were literally making it up as they went along; both the code itself, and the ways to design and engineer code, e.g., using flow charts.

Then computers got smaller and more powerful, applications became either more widespread or greater in scope, or both. Software engineering got more professional. Design methodologies for large and complicated engineering projects were produced, e.g., the traditional waterfall model for software development. The difference between functional and non-functional requirements was recognized, i.e., that the software had a function to perform, but that there were also criteria against which the performance of that function could be evaluated. The nascent software houses stopped using aptitude tests to assess suitability for hiring programmers, and recognized that having a university degree in computing or computer science constituted a sufficient qualification for the job.

As the applications became more widespread, though, the programmers became suppliers to the users, and the interface was provided by the same programmers, not always successfully [1], [2]. Usability and user-friendly became buzz-words, human-computer interaction became a subject in itself, and design methodologies were produced which were intended to ensure that software engineers delivered products that were fit-for-purpose, by involving end users actively in the design and development process, e.g., participatory design, or by ensuring that systems were targeted at their users rather than forcing users to adapt in order to accommodate the system, e.g., user-centred systems design. Functional requirements had to consider using the computer or device to support achievement of a function that the user had to perform, and non-functional requirements had to take into account criteria against which to measure that performance, i.e., usability was defined (partially) in terms of achieving quantifiable performance levels with acceptable satisfaction and “cost.”

Eventually, of course, computers became sufficiently small, cheap, and powerful to enable ubiquitous computing, pervasive computing, and the Internet of Things. Thus was enabled the digital transformation, the transformation of commercial processes and organizational structures through the use of digital tools and technologies, although the same transformation is impacting all forms of infrastructure: access to news, information, and entertainment; access to education, water, energy, medical treatment, and transportation; access to systems of justice, governance, and political engagement; access to communities, communal resources, and control over the local environment.

However, as early as the 1990s, it was recognized that it was non-obvious how to design digital systems that were supported by this infrastructure: relationships, experiences, priorities, norms, institutions, relational economies and economies of esteem, morals, and ethical decisions — the things that people actually care about. Designing for the maintenance or sustenance of these qualitative human values as supra-functional requirements was the basis of and motivation for value-sensitive design, a design...
methodology that attempts to put human values as primary requirements of the design process (3). This methodology has been most clearly instantiated in Privacy by Design (4), which specifies design principles predicated on respecting and assuring privacy being a systems developer's fundamental mode of operation. In the same vein, most recently, Democracy by Design has been proposed, specifying design principles which should be observed by developers of socio-technical systems intended to support collective self-governance (5).

This is a two-way street though. If digital technologies can be designed to maintain or sustain values, then they the same technologies can be designed to manipulate or undermine those same values. For herein lies the fundamental difference between “conventional” tools and technologies and digital ones: using a knife does not affect the mindset or values of the cutter; using a social media platform (say) can. Using a knife does not transmit information about what was cut, where, when, and by whom; using a “smart” phone app can transmit information about where, when, and by whom used, and much more besides (why does a ride-hailing app need access to your camera, photos, and contacts?). Therefore, it is perfectly plausible for a company to specify to a software developer: “build a system that will increase the loyalty of our customers.” Moreover, it is not a great leap from there to “build a system that will increase the addiction to this game/gambling machine,” or, with enough data and (given that people are nowhere near as exceptional as they’d like to think) some simple logistic regression “build a system that will sway the voting patterns of this segment of the electorate.”

Given these requirements, how is the practice of software engineering meeting the challenge of delivering tools and technologies for the digital transformation? There are still monopolistic providers dominating certain domains (e.g., eCommerce — Amazon; Internet search — Google; social media — Facebook, etc.); there are still “large” software houses producing applications for “large scale” industrial, commercial, and public sector clients; there are any number of coordinated peer production systems based on open source software (6); and there is a burgeoning cottage industry churning out apps in the hope that one them will “go viral.”

Generally, this represents some significant changes in the way code is being produced. For one, there seems to be a tendency to start from the empty program and debug until it gets past the compiler. There also seems to be a strong tendency to write “glueware”: the programmer just downloads modular components and writes code that “glues” the functionality together. There is also a global software market that lends itself to unscrupulous development, for example, all the Macedonian websites generating revenue from automated advertising engines through the dissemination of “fake” or sensationalist news (7). Finally, there is a particularly strong tendency to see the users as nothing more than an aggregated revenue stream, and to design systems that create lock-in or use psychological manipulation to retain attention (8). Combined with a lack of critical thinking, this being underdeveloped in educational systems focusing on league tables and performance targets, and a buy-in to corporate culture in order to get ahead, there is no questioning of policy, especially of policy from the users’ perspective: did it really not occur to anyone to ask if it was a good idea to put a microphone in everyone’s kitchen, hook up the data stream to a warehouse sized computer, mine the stream using sentiment analysis, link it to a social media account, and then push advertising to that account?

Essentially, though, there seems to be a reversion to the condition identified at the beginning of this article: we are seeing a lack of professionalism, or professional training, in the development of software. For example, a member of the IEEE or ACM supposedly subscribing to the institution’s Code of Ethics would not, presumably, produce a platform knowingly disseminating false or malicious information. Critically, though, this is a lack of concern for values, specifically the value of everyone at least agreeing on the same set of “facts.” Similarly, at the core of disrespect for users’ privacy is a lack of concern for values. Underpinning all of this is a lack of concern for design: participatory, user-centred, and especially value-sensitive design. The developers of digital technologies are not only not asking users about their requirements or values, they are not even asking themselves what kind of societies or cultures their innovations are creating, that they would have to live in themselves (although some tech-giant CEOs seem to have a very different conception of privacy for themselves than they have for the rest of us).

The same lack of professionalism and the same lack of user consultation can be observed in many other areas. To give one concrete example, in a (nameless) London college, cameras have been installed in every lecture theatre. The cameras are always on, and automatically start recording on the hour, and every hour. Lecturers have to take a specific decision to opt out — in a previous system, the lecturer had to opt in. The justification for this, such as it is, is nearly always, and only: the students like it, as the lectures can be used for revision.

As a rule of thumb, one could identify an implausible defense when it becomes: “we were doing it for the kids” (see the Dover Trial of intelligent design as a scientific
theory [9]: the last refuge of defense for the intelligent design advocates after their case had been systematically demolished was “we were doing it for the kids”). If the intention really was to provide revision materials, there are other ways to build such a resource without live video recording of every lecture, i.e., without intrusion into a personal space.

Another objection to this practice is based on the definition of a lecture. For some, a lecture is a real-time development and exposition of an idea involving a “multi-logue” (multiple dialogues between the lecturer and the students. It is not just the lecturer talking: a good lecturer “reads” the room and senses when s/he is (or is not) being understood by the audience. It is not necessarily a commodity that can then be sold as part of an online course. Furthermore, this is a private conversation, and designed for live delivery; a lecture designed for public exposure through a digital technology would use different expression, pace, mannerisms, spontaneity, supporting materials and so on. Just as one would not make a film of a book by videocopying each of the pages, one does make a film of a lecture by putting a camera in a lecture theatre. There is the additional burden on lecturers to be their own editors and curators of their recorded material — although the benefits to the employers and to the employees are almost entirely asymmetric [11].

Ultimately, this is the problem: people who are a) untrained in the design of computer-supported software systems, b) unaware of human factors in this process, in particular qualitative human values, and c) not immersed in the consideration of the legal, ethical, and social implications of digital technologies, make design decisions that can have severe repercussions and unintended consequences. In the case of live video recording of lectures, these are decisions that contribute to the creeping normalization of surveillance capitalism (12) and surveillance culture (13). Therefore systems design with digital technologies should be a mandatory subject for professional engineering courses in higher education. Moreover, it is also why the papers in this special issue are a timely reminder, to involve users, to consider their requirements in the design process, and to think in terms of both usability and their values, not their exploitability as commodities in an aggregated revenue stream.

No one seems to be asking permission for these practices? — one would have thought consent would be necessary to comply with new GDPR regulations. Is being recorded doing your job just an accepted part of everyday life? — other than film and TV actors, live sports and entertainment, and elected representatives (for transparency), it is difficult to think of any other profession where this is acceptable. It is not just video, either: by extension there are many other channels that can be, and already are being, used to monitor workforce performance, although the benefits to the employers and to the employees are almost entirely asymmetric [11].

EiC’s Note

A warm welcome to Professor Franco Zambonelli (Università di Modena e Reggio Emilia) and Dr. Simon Power (Edinburgh Napier University) who join T&S Magazine as Associate Editors beginning July 1, 2018. Thank you in advance for your contributions to the magazine.

Author Information

Jeremy Pitt is Professor of Intelligent and Self-Organizing Systems at Imperial College London, U.K. Email: j.pitt@imperial.ac.uk.

References