THE PRECAUTIONARY PRINCIPLE

I read with interest “The Precautionary Principle in a World of Digital Dependencies” (W. Pieters and A. van Cleeff, June 2009, pp. 50-56). While the precautionary principle is a concept strictly used for regulation policies when fears for irreversible health and safety hazards exist, the authors use arbitrary arguments to transform it to a legal obligation of software engineers to protect users’ morality.

The authors struggle to justify the application of the principle to IT by inventing the concept of “digital irreversibility” and making an amusing analogy of the “infosphere” to nature. Based on this, they then make the even more arbitrary mental leap to apply the principle from cases of irreversible harm to natural resources to cases of mere damage to private digital assets, such as copyright infringement, offering the Napster case as an example. Some paragraphs later, the precautionary principle is arbitrarily redefined again, merely as “precaution against unintended and undesirable use.”

Making yet another arbitrary assumption, namely that the irreversible damage in question can pertain to human moral health, the precautionary principle is redefined again as “accountability for impacting people’s intentions.” Now software engineers not only are accountable if “their design invited someone to do something wrong,” but must also act as the moral safeguards of our society as a whole, inviting morally sound use and inhibiting undesirable or controversial actions through their software.

Note the dangerous transition from the original principle of preventing irreversible harm to public health or nature, to a moralistic (and, as proposed by the authors, legal) imperative of not only preventing (yet unfounded and by definition subjective) “harm” to society’s morality, but also of inviting morally sound use.

While it is founded on arbitrary arguments and is unscientific, since it is contrary to natural hazards, which can be scientifically (dis)proved, whereas moral hazards cannot, the authors’ proposal also faces obvious practical difficulties and risks associated with our globalized technology because moral standards differ significantly among and within societies.

But I have a more fundamental objection. By becoming an engineer, I learned that my moral obligation is to advance other people’s well-being by solving their practical problems. Had I wanted to protect their souls, I would have become a preacher. And even then, I would not do that against their free will.

Gregory Farmakis
gregory.farmakis@agilis-sa.gr

The authors respond:

Gregory Farmakis’s reaction to our article shows precisely why we should at least think about precaution in software engineering. Mr. Farmakis assumes that technology is morally neutral and merely a means to solve people’s problems. A century of studies of science, technology, and society have shown that this view is problematic. We can hardly say that the mobile phone is merely a device that solved an urgent problem; it has profoundly changed people’s experience and behavior. The same holds for social networking sites. An engineer who solves people’s practical problems often also “solves” many other issues for them. The solution will both increase and limit people’s freedom, and the trick is to limit their freedom in the right way.

Application of the precautionary principle, then, is a way to treat society with the same respect as nature. It is meant to provide early identification of possible side effects, so that they can be tested against prevailing moral values. Societies of engineers already have ethical codes, and the precautionary principle may be a way to make these more proactive, if they are not already. It is insufficient to reject such application by arguing that it is unscientific. The precautionary principle, even in health and safety, deals with lack of scientific certainty in the first place. It is not of principal importance whether this uncertainty is about nature or morals, although the goals may be less obvious in the latter case.

We have made clear in the article that we do translate the principle to a new area, with all the risks involved. If the translation is imperfect, we are happy to accept suggestions for improvements. This, however, should not be a reason to reestablish an instrumental view of technology and ignore opportunities for moral improvement by stating that engineers merely solve practical problems.

DEFINING ENGINEERING

In “Agility and Respect” (The Profession, July 2009, pp. 100, 98-99) Neville Holmes adopts an idiosyncratic definition of engineering in order to show that programmers are technicians.

If engineers look after people’s interests, while technicians look after the engineers’ interests, then none of the designers of IEEE Spectrum’s recent “25 Microchips that Shook the World” or of the machines that made them, would rate as engineers. Of course, they
indirectly helped nonengineers, but so do programmers.

In the common view, engineering applies science—explicitly looking after people’s interests is a specialization thereof. Engineers differ from technicians because they exercise judgment of a sort that cannot be codified into rote procedures and standards. Programming has proven highly resistant to this sort of reduction, and I think this is why there is so much variability in programming ability.

If software engineering cannot let go of programming, it is not from want of trying. When it was introduced to commercial data processing, there was much focus on requirements specification and testing, but for actually making the product, it offered little more than design rules of thumb and coding standards. The results were disappointing and, in reaction, the Agile Manifesto, derived partly from observing successful teams, stepped into the gap.

We are told that software engineering’s programming fixation is why it lacks credibility with engineers, but I suspect the foremost issue is the lack of rigor. Engineers are expected to know the relevant underlying science and technology, and, to put it simplistically, there is a lot of math—an observation indicative of a rigorous underlying theory. This is expected even of engineers in people-oriented roles so they can exercise informed judgment.

Perhaps anyone claiming the title of software engineer should demonstrate a competence in formal methods. I suggest this neither for respect, nor because I think we should use formal methods extensively, but because knowing what rigorous software development demands informs how we go about it less formally. This is so even for Mr. Holmes’s putative data engineers and technicians to work in close and continuing partnership. The challenge to computing professionals of all kinds is to respect program technicians and to work in partnership with them on major projects.

Of course, computing technicians of various kinds are needed. Many already exist and work on “domestic” problems, just as many electricians do. What is missing is the corps of specialist programmers to work with software engineers, along with the intense technical training for such programmers.

We welcome your letters. Send them to letters@computer.org. Letters are subject to editing for style, clarity, and length.