DOI:10.1145/2566966

Arvind Narayanan and Shannon Vallor

Computing Ethics

Why Software Engineering Courses Should Include Ethics Coverage

Encouraging students to become comfortable exercising ethical discernment in a professional context with their peers.

Software developers create the architectures that govern our online and increasingly our offline lives—from software-controlled cars and medical systems to digital content consumption and behavioral advertising. In fact, software helps shape, not just reflect, our societal values.[a] Are the creators of code aware of this power and the responsibilities that go with it? How, and to what extent, are they trained in the ethics of their discipline?

Like medical, legal, and business ethics, engineering ethics is a well-developed area of professional ethics. In 2000, the organization that accredits university programs and degrees in engineering (ABET) began to formally require the study of engineering ethics in all accredited programs.[b]

Yet most engineering ethics textbooks focus primarily on ethical issues faced by civil, mechanical, or electrical engineers. The case studies they typically include—the Challenger explosion, the Ford Pinto fires, the Union Carbide/Bhopal disaster—depict harms caused by ethical lapses in those fields. Of course, the cars and rockets and bridges built today depend upon critical software for their safe operation, and failure of these software systems can result in death or grievous injury. However, the distinctive ethical dilemmas that arise in the software engineering context are not yet being sufficiently addressed.

In the Internet era, the software development and deployment process has some peculiarities that exacerbate the ethical issues for software engineers. First, the shortened life cycle has weakened and in some cases obliterated software review by management and legal teams. So software engineers may deploy code directly to end users—in stark contrast to, say, a civil engineering project with a years- or decades-long life cycle and multiple layers of oversight. For Web applications such as Facebook, individual engineers or small groups of

a See http://www.nyu.edu/projects/nissenbaum/vid/about.html

b See http://www.abet.org/engineering-change/

CREDIT TK

engineers code and deploy features directly, and indeed the culture takes pride in this. Even where more traditional development practices prevail, some deployments like bug fixes are shipped with only technical (and not ethical) oversight. In such circumstances, the individuals deploying the code may have to rely on their own familiarity with ethics when faced with the old question: it may be legal to do this, but *should* we?

Second is the issue of scale—perhaps the defining feature of the software revolution. For software, the entire world is typically part of the addressable market. This scale creates the potential for individual software engineers to produce great good, but with it naturally comes the ability to cause great harm, especially when combined with the ability to deploy code directly to end users. Here is a benign but illustrative example. On June 9, 2011, Google released a "doodle" honoring Les Paul, which users found addictive to play with. This is a type of project that is typically done by an individual engineer on his or her "20% time," in a day or two. A third party, RescueTime, later estimated that 5.3 million hours were ultimately spent by people playing this game.[c]

Consider that 5.3 million hours equates to about *eight lifetimes*. Did the doodle make a positive contribution to the world? Do engineers at Google have an obligation to consider this question before releasing the feature? What principle(s) should they use to determine their answer about the benefits and/or harms of their work? Often, individual software engineers must grapple directly with such issues, instead of relying on management or anyone else.

Finally, software engenders ethical concerns other than concrete harms. Software embeds moral and cultural values and inevitably nudges society toward these values. Today's Web and information services are designed around the centralized collection and control of personal data. One effect is that social interactions happen more often in public view; another is the changing balance of power between users, companies, and governments. Further, the lack of geographic constraints means software engineers are generally unfamiliar with the culture and values of many of the end users of their products. Cost cutting often leaves little room for user studies or consultations with experts that might allow software development firms to acquire this familiarity.

Nevertheless, software engineers share with everyone a desire to flourish and do well in life and work. Thus, ethical obligations have a professional *and* a personal dimension. Without a sense of personal ethics, a software engineer would be indifferent to her actions' effects on the lives of others in circumstances not explicitly addressed by a professional code of ethics. But for professionals whose work impacts public welfare, personal ethics is not enough. Without a sense of professional ethics, individuals might justify to themselves conduct that would be much more difficult to justify in front of others. Additionally, professional ethics help us understand how ethical standards and values apply to a particular type of work. For example, what does integrity look like in a software engineering context? What sort of specific coding practices demonstrate integrity, or a lack of it? This is something that professional codes of ethics—and discipline-specific ethics training—can help students learn to see.

Being a professional means being a part of a moral community of others who share the same profound responsibilities. Embedding coverage of ethics in software engineering courses would help students draw strength and wisdom from dialogue with other future members of their profession—colleagues who will face the same types of moral dilemmas, struggle with the same sorts of tough decisions, and ultimately seek to earn, in similar ways, the respect of their peers and the broader public. Software engineering professors are in the best position to spark that dialogue. So why is not applied ethics currently taught much in computer science and software engineering courses? We cannot know for sure, but there are some plausible reasons.

Software embeds moral and cultural values and inevitably nudges society toward these values.

First, the algorithmic and engineering techniques students learn are fantastically general (all that computers do, at an underlying level, is manipulate ones and zeroes). The flip side is that computer science is generally taught in a manner divorced from practical context. This abstraction makes computer science education powerful; it would be silly to train students to work specifically in the music or enterprise software industry. However, the exclusive focus on general techniques leaves students with the impression the implications and consequences of their work product are not their proper concern. When such students graduate to solving problems in the real world, they are likely to adopt the type of thinking that prevails in many parts of the industry—that anything technically feasible is fair game, and that ethical issues are best handled by compliance teams and Terms of Service documents.

Second, these technologies are inherently morally ambiguous. For example, the same digital signatures that make the lock icon appear in a browser (indicating you are not connected to a spoofed site) are used by malware authors to ensure zombie machines obey only commands from their true bot-masters. Or consider machine learning and collaborative filtering systems used to recommend new bands or books you might like, or to detect credit card fraud: inevitably, these systems introduce systematic biases into our patterns of consumption and behavior. The so-called "filter bubble" arises when algorithmic systems, such as Google search or the Facebook news feed, decide what information to show a user based on his or her past pattern of searches and clicks.[d] The worry is

c See http://blog.rescuetime.com/2011/06/09/google-doodle-strikes-again/

d See http://www.amazon.com/Filter-Bubble-What-Internet-Hiding-ebook/dp/B004IYJE6A/

that users will be fed reinforcing viewpoints and find themselves isolated in a personalized "echo chamber."

At the level of demographics, the seemingly fair principle of treating "similar" users similarly can lead to deepening disparities. Online ads have been shown to display racial bias,[3] and prices online have been shown to vary based on users' personal attributes.[4] In the political sphere, public-interest groups have been investigating the implications of campaign messages tailored to the individual.[2] Perhaps most worryingly, when systems with direct power over our lives, such as the no-fly list, use opaque machine-learning based techniques to make decisions, we lose the safeguards of due process.[1]

Thus, when confronted with the bewildering variety of ethical questions that may arise from a single technology, engineering professors might well prefer to leave the whole topic to some "ethics professionals."

A third factor is it is often unclear how coding practices might mitigate these harms and risks. In all likelihood, for example, the racial bias of online ads was not the result of explicit intent by engineers but rather an emergent property of a system aiming to maximize the click-through rate. Even defining the notion of fairness of an algorithmic system in a mathematical way that computers can interpret remains elusive—let alone a general procedure for designing systems that satisfy this goal.[e]

But these are all reasons for teaching ethics in software engineering courses, not ignoring it. Ethical judgment, what philosophers often call *practical wisdom*, is needed most when moral dilemmas arise for which there are no easy solutions. And the software engineers are often the only ones who fully understand both the benefits and the dangers engendered by new technologies.

How to teach software engineering ethics? One choice is between a separate ethics course versus integrating ethics discussion into every course. Both approaches are valuable, but the latter is perhaps more immediately useful. We propose that computer science educators include a discussion of ethics with every significant technology they teach. Hypotheticals and case studies are two powerful and complementary tools for this purpose. Hypotheticals allow students to quickly isolate important ethical principles in an artificially simplified context; for example, one might ask students what ethical principles or values come into play if a manager suggests promising a customer 'fictionware'—a desirable feature that is actually impossible to develop or deliver. Discussions of case studies, on the other hand, allow students to confront the tricky interplay between the sometimes competing ethical values and principles relevant in real-world settings. For example, the Google Street View case might be used to tease out the ethical conflicts between individual and cultural privacy expectations, the principle of informed consent, Street View's public value as a service, its potential impact on human perceptions and behaviors, and its commercial value to Google and its shareholders. Then, to bring students back to the practical space of ethical action, the professor might pose a realistic hypothetical, asking students to explain and defend how, as a Google project manager, they would evaluate a proposal to bring Street View technology to a remote African village. What questions should be asked? Who should be consulted? What benefits, risks and safeguards considered? What trade-offs weighed?

Students should be in the habit of considering how the code they write serves the public good.

What matters in these exercises is not that students can arrive at the 'right' answers; nor even that the instructor have them in hand. In many real-life cases there is no single right answer, only a range of more or less ethically informed and wise responses. What matters is that students get comfortable exercising ethical discernment in a professional context alongside their peers.

Educators should also seek to instill professionalism in students; currently many software engineers seem indifferent to or even actively reject this aspect of their work. Collaborative activities could help reinforce the sense of belonging. For example, students could be tasked with doing a collective survey of ethical lapses in the software industry, along with a survey of ethical attitudes among employees of various companies. Since these companies are prospective employers, the results will be of immediate value to students, increasing their motivation.

Habits are powerful: Students should be in the habit of considering how the code they write serves the public good, how it might fail or be misused, who will control it; and their teachers should be in the habit of calling these issues to their attention. Students may resist a bit—after all, one of the things that draws some people to programming is the opportunity to retreat into a happy zone of bit twiddling detached from the world, and we are proposing to habituate them out of this. Other students, however, may be more drawn to such an approach.[f] Between confronting and evading ethics, confronting ethics is the only defensible choice. C

e See C. Dwork et al., Fairness Through Awareness; http://arxiv.org/abs/1104.3913

f See http://www.smartplanet.com/blog/bulletin/computer-science-is-for-women-too/

References

1. Citron, D.K. Technological due process. *Washington University Law Review 85* (2007), 1249–1313; http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1012360.
2. Larson, J. Message machine starts providing answers. *ProPublica* (Oct. 18, 2012); http://www.propublica.org/article/message-machine-starts-providing-answers.
3. Sweeney, L. Discrimination in online ad delivery. *Commun. ACM 56*, 5 (May 2013), 44–54; doi>10.1145/2447976.2447990.
4. Valentinio-Devries, J., Singer-Vine, J., and Soltani, A. Websites vary prices, deals based on users' information. *The Wall Street Journal* (Dec. 24, 2013); http://on.wsj.com/19pz4S5.

Arvind Narayanan (arvindn@cs.princeton.edu) is an assistant professor of computer science at Princeton University, NJ.

Shannon Vallor (svallor@scu.edu) is an associate professor of philosophy at Santa Clara University, CA.

The authors thank Irina Raicu for her feedback and edits. The authors are also grateful for the comments received from numerous people including the anonymous reviewer and David Robinson.

A portion of this article appears in the introductory section of "An Introduction to Software Engineering Ethics"—a concise curriculum module available at no cost, to all professors, from the Markkula Center for Applied Ethics at Santa Clara University.