DOI:10.1145/2018396.2018407

Michael Davis

Computing Ethics
Will Software Engineering Ever Be Engineering?

Considering whether software engineering and engineering can share a profession.

An anonymously attributed adage states: "With another name, social engineering would not be mistaken for engineering." Approximately 15 years ago, I published a short article in the *Journal of Engineering Education* arguing—among other things—that software engineering was not then engineering.[1] I have now been asked whether enough has changed to make me think software engineering is engineering. My answer is: much has changed—with some changes weakening the separation between engineering and software engineering and some reinforcing it—but, overall, the argument stands. This answer will surprise those who, unaware of that article, think software engineering's status as engineering is obvious. I therefore think it wise to precede any explanation of why software engineering is not engineering by disposing of a few unexamined presumptions that might make software engineering's status as engineering seem obvious.

The Software Engineering Code of Ethics and Professional Practice differs in significant ways from all the engineering codes I know.

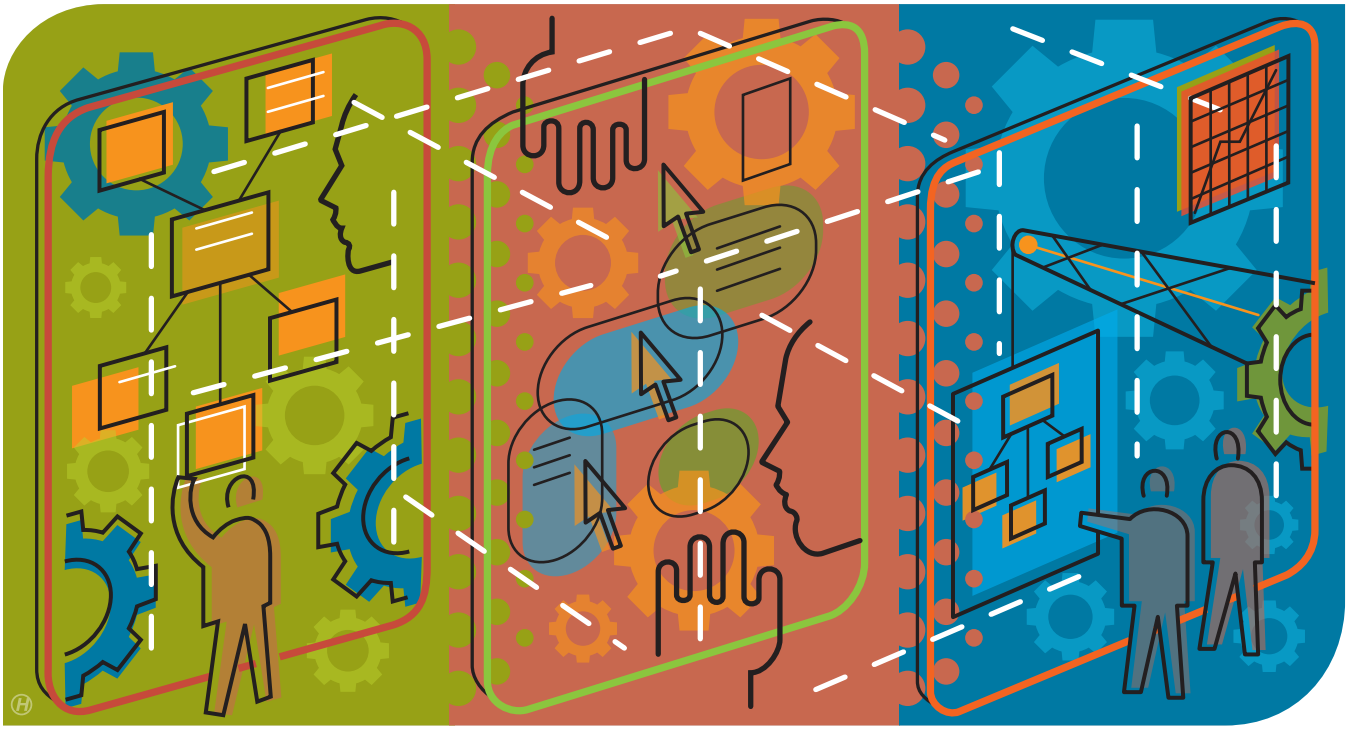
Senses of Engineering

"Engineering" has at least four senses in English. One, the oldest, understands engineering as tending engines (originally, "engines of war"). Casey Jones was an engineer in this sense; so is the custodian of my building, a licensed "boiler engineer"; and so too, the sailor rated "marine engineer." Neither engineers (strictly speaking) nor software engineers are engineers in this sense.

Almost the opposite of this first sense is what we might call the functional sense, engineering-as-invention-of-useful-objects. In this sense, the first engineer may have been the caveman (or cavewoman) who invented the club, cutting stone, or fire pit. Though this sense would certainly make software engineers engineers, there are at least two reasons to reject it here. First, the functional sense is too broad. Architects, industrial designers, and even weekend inventors are all engineers in this sense, making software engineering's claim to be engineering uninteresting. Second, the functional sense is anachronistic. It takes a sense of "engineering" that did not exist much before 1700 and applies it to cavemen, carpenters, tinkerers, and the like, who would have understood themselves quite differently.

The functional sense of engineering nonetheless seems relevant here. Software engineering's official Body of Knowledge offers this definition of software engineering: "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software."[2] The Body of Knowledge assumes, without argument (a mere "that is"), that engineering is a certain function, any "systematic, disciplined, quantifiable approach to the development, operation, and maintenance [of something]". That assumption must be false. It would force us, for example, to rank accounting—a field no one supposes to be engineering—as "financial-records engineering" (since accounting is a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of financial records).

Closer to our subject is a third sense, engineering-as-discipline. A discipline is a distinctive way of carrying on an activity, some combination of knowledge, skill, and judgment that must be



learned. Any craft or trade has its discipline—as do many activities that are not craft or trade, such as meditation or calisthenics. In this sense, neither architects, nor industrial designers, nor weekend inventors are engineers. Architecture and industrial design each have a discipline easily distinguished from engineering's. Weekend inventors have no discipline at all; they may invent any way they like.

Software engineering is not engineering in this third sense. The body of knowledge engineers are supposed to learn differs in important ways from software engineering's body of knowledge. So, for example, engineers have to take courses concerned with the material world, such as chemistry and statistics; software engineers do not. Software engineering's official Body of Knowledge was in fact an important step in clarifying the distinction between engineering proper and software engineering. It requires software engineers to know things other engineers do not and not to know some things other engineers do know.

Software engineering has, indeed, become a profession. What it has not become is part of the engineering profession.

The last sense of engineering we need to distinguish here is engineering-as-profession. A profession is (we may say) a number of individuals in the same occupation voluntarily organized to earn a living by openly serving a moral ideal in a morally permissible way beyond what law, market, morality, and public opinion would otherwise require.[a] An occupation is a discipline by which one may, and some do, earn a living. Both engineering and software engineering are now occupations but, having (as just noted) different disciplines, must be different occupations. That is one reason why they cannot share a profession. There is another.

The Software Engineering Code of Ethics and Professional Practice differs in significant ways from all the engineering codes I know. Software engineers are, for example, supposed to "[m]oderate the interests of the software engineer, the employer, the client and the users with the public good" (1.02).[b] Engineers do not now have such a duty to moderate.

Software engineering has, indeed, become a profession. What it has not become is part of the engineering profession. Anyone who claims otherwise must find a sense of engineering different from those distinguished here, one that makes software engineering a part of engineering without including as well disciplines, occupations, or professions, such as architecture or accounting, that clearly are not part of engineering.

Professions are voluntary associations. You cannot become a member simply by claiming to be one. You must be admitted (by the profession, not just by a technical society like the ACM). Engineering has a long history of other occupations claiming to be engineering: recent examples include genetic engineering (a kind of tinkering with genes); reengineering (a fad in management); and financial engineering (gambling on Wall Street). Software engineering actually began with an attempt to copy engineering practices, making its claim to be engineering more respectable

a For a defense of this definition, see my article "Is Engineering a Profession Everywhere?" *Philosophia 37* (June 2009), 211–225.

b Software Engineering Code of Ethics and Professional Practice (1999); http://www.acm.org/about/se-code. For history of this document, see my essay "Code Writing: How Software Engineering Became a Profession," Center for the Study of Ethics in the Professions, Chicago, 2007; http://hum.iit.edu:8080/aire/sea/1/book/index.html.

ILLUSTRATION BY HANK OSUNA

than most. But the enormous complexity of software has forced software engineering to develop in ways engineering has not—and may never.[c] Many of the very methods that make software engineering useful distinguish it from engineering. Engineers have good reason to continue to treat software engineers as belonging to another profession.[d]

I have, I hope, just explained why I still think software engineering is not engineering in a way that engineers should recognize. I now want to point out four reasons to think that engineering might someday merge with software engineering. All four are, oddly, changes in engineering, not software engineering.

▸ Electrical and computer engineering (ECE) is often thought to be the field of engineering closest to software engineering. Over the last decade, ECE has become less committed to traditional engineering courses concerned with the material world. So, for example, a number of ECE departments, including the one at the University of Illinois at Urbana-Champaign, have stopped requiring statics, dynamics, and thermodynamics. If that trend continues, then either ECE will split off from the main body of engineering or engineering's core of required *engineering* courses will increasingly resemble software engineering's.

▸ Since the 1700s, engineers have had to know just two natural sciences: physics and chemistry. Recently, some programs in environmental engineering, biomedical engineering, and agricultural engineering have begun to allow students to substitute biology for physics or chemistry. For engineers, this makes sense, since several of the new frontiers of engineering rely on biology rather than physics and chemistry (as until recently). But, if this trend continues, engineering's *science* core will increasingly resemble the science courses software engineers take to satisfy general distribution requirements.

Whether or not software engineers do any engineering, engineers increasingly engage in activities that look like software engineering.

▸ Engineers are increasingly replacing mechanical systems with software. Not only do most engineers now use software regularly, many write specifications for software, modify existing programs themselves, or even write (simple) programs. Whether or not software engineers do any engineering, engineers increasingly engage in activities that look like software engineering (even if these engineers do not call themselves "software engineers" and do not work the way that software engineers would). Whether some fields of engineering will dissolve into software engineering seems an open question.

▸ Computer science used to have an accreditation body separate from engineering's. That is no longer true. All computer science programs, including software engineering, are now under engineering's accreditation body, ABET. Of course, the accreditation process and standards distinguish between engineering programs and computer science programs. But that distinction does not preclude eventual merger. ABET has always distinguished between various fields (or subdisciplines) of engineering. So, for example, it always sent mechanical engineers to review a mechanical engineering program; electrical engineers, to review an electrical engineering program; and so on. The expansion of ABET's accreditation powers makes it easier than before for software engineering to merge into engineering, indeed, for all of computer science to do that.

Having pointed out four reasons that seem to point to software engineering's eventual merger with engineering, I now point out three reasons to believe the merger will not happen soon, if at all:

▸ All engineering is still fundamentally about physical systems; software engineering is not. Even a field so closely allied to software engineering as computer engineering must take into account physical factors in design, for example, heat produced in a microchip or speed of electrical current, to a degree software engineers do not.

▸ Software engineering is today a large profession, indeed, one of the largest—half the size of engineering, true, but about the same size as medicine or law. With so many practitioners, software engineering is more likely to divide than to join up with another large profession.

▸ If computer science ever ceased to be the home of software engineering, the most likely new home might well be management information systems or information technology management. These business disciplines resemble software engineering at least as much as engineering does. In practice, most software engineers work more with information systems managers than with engineers.

Conclusion

Whether knowledge of the future is possible is a perennial question in philosophy. What is certain is that prophets are seldom right on any important question. So, I make no claim to know whether software engineering will ever merge with engineering. I claim only to know that—despite the common term "engineering"—software engineering is not now engineering. C

c Michal Young and Stuart Faulk, "Sharing What We Know About Software Engineering," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research* (FoSER '10), ACM, 439–442, argue that engineering has much to learn from software engineering—inadvertently making clear how much engineering's discipline differs from software engineering's.

d For a darker route to this conclusion, see David L. Parnas, "Risks of undisciplined development," *Commun. ACM 53*, 10 (Oct. 2010), 25–27. Note that Parnas, though a star of software engineering, is an electrical engineer—both by discipline and declaration—looking at software engineering the way knowledgeable engineers typically do.

References

1. Abran, A. et al. *Guide to the Software Engineering Body of Knowledge—2004 Version*. IEEE Computer Society (2004), 1.
2. Davis, M. Defining engineering: How to do it and why it matters. *Journal of Engineering Education 85*, (Apr. 1996), 97–101.

Michael Davis (davis2643@gmail.com) is a senior fellow at the Center for the Study of Ethics in the Professions and a professor of philosophy at Illinois Institute of Technology, Chicago.

Thanks to Keith Miller and Rachelle Hollander for helping me think through this column.