3

Are "Software Engineers" Engineers?

> Today, the field has emerged as a true engineering discipline."
>
> John J. Marciniak, "Preface," *Encyclopedia of Software Engineering* (1994)

> If you are a " engineer," you could be breaking the law. It is illegal in 45 states to use that title, warns Computerworld newspaper. People who aren't educated and licensed in 36 recognized engineering disciplines can't call themselves "engineers," and computer professionals often don't qualify.
>
> *Wall Street Journal,* June 7, 1994, p. 1.

For those interested in professions, the emergence of what may be a new profession should generate the same excitement that the discovery of a new class of objects in the sky generates among astronomers. Not only is it inspiring to watch, it is a chance to put theory to work in unexpected ways, a chance to separate the charming from the true. This chapter begins with the emergence of software engineering as a distinct discipline, occupation, and, perhaps, profession.

The term "software engineering" came into currency after a 1967 North Atlantic Treaty Organization conference on software design and testing used that term in its title.[1] Today, thousands of people are called software engineers, do something called software engineering, and have sophisticated employers willing to pay them to do it.[2] Yet, software engineering is no ordinary engineering discipline. Few software engineers have a degree in engineering. Some are graduates of a program in computer science who had a single course in "software engineering." (Typically, that course is taught by someone with a degree in computer science rather than engineering.) Most software engineers are programmers with no formal training in engineering.[3] Are software engineers nonetheless engineers? What, if anything, makes this question worth answering?

Let me answer the last question first: Defining a field is more than semantics. How we define a field can affect how it develops. Software engineering may be a field whose progress is threatened by the analogy with engineering, a field pushed toward an unnecessarily rigid curriculum.[4] That is the first reason our questions

about software engineering are worth answering. Second is that trying to answer them will help us understand engineering. What are its boundaries? What is at stake when we draw such boundaries? A third is that trying to answer such questions tests the utility of our history of engineering. What insight can this history give us?

The insight may be disappointing. What I show is that we can't tell whether software engineering is engineering. Only the future can tell. The best we can get from engineering's history is insight into why that is so. Getting that insight leads me to defend two theses: First, that software engineers are not engineers merely because they do much that engineers do or know much that engineers know; second, that whether software engineering can or should be a field of engineering depends on whether software engineers can or should be educated in the way engineers are. These two theses rest on a third: Engineering is (or, at least, should be) defined primarily by its curriculum rather than, as we might expect, by what engineers in fact do or know. Because the defense of the other two theses rests on the third, it is with the third that we must begin.

The Standard Definition of Engineer

The standard definition of engineer is something like this: An engineer is a person who has at least one of the following qualifications: (1) a college or university B.S. from an accredited engineering program or an advanced degree from such a program, (2) membership in a recognized engineering society at a professional level, (3) registration or licensure as an engineer by a government agency, or (4) current or recent employment in a job classification requiring engineering work at a professional leve1.[5] The striking feature of this definition is that it **presupposes** an understanding of engineering. Three of the four alternatives actually use the term "engineering" to define engineer; and the other, alternative (3), avoids doing the same only by using "as an engineer" instead of "to practice engineering."[6]

This definition and others like it are important. They determine who is eligible for admission to engineering's professional societies, who may be licensed to practice engineering, and who may hold certain jobs. Such definitions are also eminently useful. For example, they help the Census Bureau exclude from the category of engineer drivers of railway engines, janitors who tend boilers in apartment buildings, and soldiers wielding shovels in the Army's Corps of Engineers. These, though still called engineer, clearly are not engineers in the relevant sense. They are engineers only in a sense now obsolete.

However, the standard definitions do not suit our purpose. They do not tell us whether a software engineer is an engineer-or even how to go about finding out. A software engineer may, for example, work at a job classified as requiring software engineering (at a professional level). That will not settle whether a software engineer is an engineer: What an employer classifies as "engineering" (for lack of a better word) may or may not be engineering in the relevant sense.[7]

What will settle the question? In practice, the decision of engineers. An organization of engineers accredits baccalaureate and advanced programs in engineering. Other organizations of engineers determine which societies with "engineer" in the title are engineering societies and which-like the Brotherhood of Railway Engi-

neers-are not. Engineers also determine which members of their societies practice engineering "at a professional level" and which do not. Government agencies overseeing registration or licensure of engineers, though technically arms of the state rather than of engineering, generally consist entirely of engineers. And, even when they do not, they generally apply standards (education, experience, proficiency, and so on) developed by engineers. Engineers even determine which job classifications require professional-level engineering work and which do not.

The standard definition settles many practical questions, but not ours. Because engineers divide concerning whether software engineering is really engineering, to say that software engineers are engineers if they engage in professional engineering is-for engineers and those who rely on their judgment in such matters-merely to restate the question.*

That is a practical objection. There is a related theoretical objection. A definition of engineer that amounts to "an engineer is anyone who does what engineers count as engineering" violates the first rule of definition: 'Never use in a definition the term being defined.' That rule rests on an important insight. Though a circular definition can be useful for some purposes, it generally carries much less information than a noncircular definition. So, for example, a dictionary that defines ethics as "morality" and then defines morality as "ethics" helps only those who understand one of the terms but not the other. The smaller the circle, the less helpful a circular definition is.

How can we avoid the standard definition's circularity? The obvious way may be to define engineering without reference to engineer and then define engineer in terms of engineering. The National Research Council (NRC) in fact tried that approach, coupling its definition of engineer with this definition of engineering:

> Business, government, academic, or individual efforts in which knowledge of mathematics and/or natural science is employed in research, development, design, manufacturing, systems engineering, or technical operations with the objective of creating and/or delivering systems, products, processes, and/or services of a technical nature and content intended for use.[9]

This definition is certainly informative insofar as it suggests the wide range of activities which today constitute engineering. It is nonetheless a dangerous jumble. Like the standard definition of engineer, it is circular: "Systems *engineering"* should not appear in a definition of engineering. The same is true of "technical" if used as a synonym for engineering. (If not a synonym, technical is even more in need of definition than engineering is and should be avoided for that reason.) The NRC's definition also substitutes uncertain lists-note the "and/or"-where there should be analysis. Worst of all, the definition is fatally overinclusive. Not only are software engineers engineers according to the definition, but so, too, are many whom no one supposes to be engineers, not only applied chemists, applied mathematicians, architects, and patent attorneys but, thanks to the and/or between mathematics and natural science, even actuaries, accountants, financial analysts, and others who use mathematics to create financial instruments, tracking systems, investment reports, and other technical objects for use.

Though much too inclusive, this definition of engineering shares with most others three characteristic elements. First, it makes mathematics and natural science central to what engineers do." Second, it emphasizes physical objects or physical systems. Whatever engineering is, its principal concern is the physical world *rather than* rules (as in law), money (as in accounting), or even people (as in management). Third, the definition makes it clear that, unlike science, engineering does not seek to understand the world but to remake it. Engineers do, of course, produce knowledge (for example, tables of tolerances or equations describing complex physical processes), but such knowledge is merely (or, at least, primarily) a means to making something useful."

Those three elements, though characteristic of engineering, do not define it. If they did, deciding whether software engineers are engineers would be far easier than it has proved to be. We could, for example, show that software engineers are not engineers simply by showing that they generally do not use the natural sciences in their work. That many people, including some engineers, believe software engineers to be engineers is comprehensible only on the assumption that these three characteristics do not define engineering (except in some rough way). But if they do not define engineering, what does?

Before answering, I describe three common mistakes about engineering to be avoided in any answer. While these mistakes may seem far from software engineering, they bring us to the best point for understanding the relation between software engineering and engineering proper.

Three Mistakes about Engineering

The NRC's definition of engineering uses "technical" twice, once as a catch-all ("or *technical* operations") and once to limit the domain of engineering ("of a *technical* nature and content"). It is the second use of technical that concerns us now. It seems to be a common mistake in usage, one even engineers make. We might summarize the mistake this way: *Engineering equals technology*.

There are at least three objections to this way of understanding engineering. First, engineering can equal technology only if we so dilute what we mean by engineering that any tinkerer would be an engineer (or, at least, be someone engaged in engineering).'* Once we so dilute engineering, we are left to wonder why anyone might want an engineer rather than some other technologist who could do the same job.[3] Why demand a software engineer rather than a programmer, software designer, or the like to do software design or development? What was the point of inventing the term "software *engineering*"?'[4]

Second, the proposition "engineering equals technology" makes writing a history of engineering (as distinct from a history of technology) impossible. The history of engineering, according to this proposition, is the history of technology. Every successful inventor is an engineer; every successful manager of industry is an engineer; and so on. We are left to wonder why our term for engineer-unlike our term for architect, mathematician, or artisan-is so recent. Why does engineering have a history distinct from technology when engineering is technology?[5] Why do engi-

neering organizations devote any effort to defining engineering? Why don't they just define technology and technologist and then say, "Ditto engineer"?

Third, "engineering equals technology" transforms talk of engineering ethics into talk of the ethics of technology. It turns professional ethics into public policy. Whatever engineering ethics is, it is, in part at least, the ethics of a profession-not merely standards governing the development, use, and disposal of technology but standards governing a certain group of technologists.

That reference to profession suggests a second mistake commonly made about engineering, one we might summarize this way: *Engineering is, by nature, a profession*. What makes this mistake attractive is the idea that a professional is a "knowledge-worker," that special knowledge defines each profession (as well as the underlying occupation). Any occupation that requires a lot of training is a profession.[6] Engineering requires a lot of training; hence, it must be a profession. Connecting profession with knowledge helps exclude from the profession of engineering those who, though they may function as engineers (or, rather, as "mere technicians"), lack the requisite knowledge to be engineers strictly so called ("engineers at the professional level"). Claiming that engineering is, by nature, a profession provides an antidote to the first mistake, but only by making another.

What is this second mistake? Thinking of engineering as, by nature, a profession suggests that organization has nothing in particular to do with profession. As soon as we have enough knowledge, we have a profession. There could be a profession of one.

Thinking this way makes much of the history of engineering mysterious. Why, for example, did engineers devote so much time to setting minimum standards of competence for anyone to claim to be an engineer? Why did they set *these* standards rather than others? Why did they suppose setting such standards relevant to being a profession? Like other professions, engineering has a corporate history that such nonprofessions as shoe repair, inventing, and politics lack. Any definition of engineering must leave room for that history. What is striking about the history of engineering-indeed, of all professions-is the close connection between organization, special standards, and claims of profession.

A third mistake may help explain the appeal of the second. We might summarize it this way: *The engineering profession has always recognized the same high standards*. There are at least two ways that this mistake is defended. One appeals to the "nature" (or "essence") of engineering. Any occupational group that did not recognize certain standards would not be engineers-or, at least, would not be engaged in engineering. Engineers have organized to set standards to avoid being confused with those who are not "really" engineers. The standards simply record what every good engineer knows; they codify rather than legislate.

The other argument for this mistake appeals to the moral nature of the engineer. It is said that engineers are always generally conscientious. To be conscientious is to be careful, to pay attention to detail, to seek to do the best one can. To do this is to be ethical. Professional ethics is being conscientious in one's work. To be a conscientious engineer is, then, to be by nature an ethical engineer.[7] Engineering societies adopt standards to help society know what it should expect of engineers,

not to tell a conscientious and technically adept engineer what to do. Informing society is, according to this view, enough to explain the effort engineers put into codes of ethics.

What is wrong with the proposition that engineering is, by nature, ethical? Like the other two mistakes, this third makes understanding the history of engineering harder. Why have engineers changed the text of their codes of ethics so often? Why do experienced engineers sometimes disagree about what should be in the code of ethics (as well as about what should be in their technical standards)? Why do these disagreements seem to be about how engineers should act, not about what to tell society?

If we examine a typical code of engineering ethics, we find many provisions that demand more than mere conscientiousness-provisions requiring, for example, engineers to help engineers in their employ to continue their education or to make public statements only in a truthful and objective manner." Such codes are less than a hundred years old.[9] Before they were adopted, an engineer only had to be morally upright and technically proficient to do all that could reasonably be expected. In those days, engineers had no responsibilities beyond what law, market, and ordinary morality demanded (and, so, had no need to inform society what to expect). The claim that engineering has always accepted the same high standards-that, for example, failing to inform a client of a conflict of interest was always unprofessional-is contrary to what we have learned about engineering.

Membership in the Profession of Engineering

As we saw in chapter 2, engineering education in the United States, almost from its beginning, had two strands: One was a series of unsuccessful experiments with various alternatives to the West Point curriculum; the other was the evolution of the West Point curriculum into the standard for engineering education in the United States. The details of that story do not matter now.[°] What does matter is that the education of engineers became more and more the province of engineering schools, and these in turn became more and more alike. For engineers, an engineer became someone with the appropriate degree from an engineering school or, absent that, with training or experience that was more or less equivalent; hence, the standard definition of "engineer" with which this chapter began.

The point of this story is not that engineering will always have the same curriculum it does today. The engineering curriculum has changed a great deal since West Point was founded in 1802; for example, there is now more calculus and less drafting. No doubt, the curriculum will continue to change. Perhaps the second year of calculus will disappear, with ecology or industrial psychology taking its place. The point of the story of engineering as I told it is, rather, that just as today's curriculum grew out of yesterday's, so tomorrow's will grow out of today's. Any new field of engineering has to find a place in that curriculum. Finding a place may mean changing the curriculum; what it cannot mean is starting fresh. Finding a place in a curriculum is a complex negotiation of social arrangements. It is like joining a family. You can change your name to Davis if you like, make yourself look like a

member of my family (perhaps even genetically), and declare yourself a member of my family, but that won't make you one. To be a member of my family, you must come in by birth, marriage, or adoption.

Some fields of engineering (for example, nuclear) seem to be born engineering, but others (mining, for example) seem to come in by the occupational equivalent of marriage or adoption. For any field not born engineering, the only way to become a field of engineering is by "marriage" (or "adoption"). Failing that, it cannot be a field of engineering. It can only don quotation marks to show irony, start another family of the same name-as railway engineering has, but without its historical justification-or choose a more suitable name.

The history of a profession tells how a certain occupation organized itself to hold its members to standards beyond what law, market, and morality would otherwise demand. The history of a profession is the history of organizations, standards of competence, and standards of conduct. For engineering in the United States, that history began after the Civil War. It is a confused story because the profession was taking shape along with the occupation. Many early members of its professional societies would not qualify for membership today.

Nonetheless, I think we can see that as engineers became clearer about what engineers were (or, at least, should be), they tended to shift from granting membership in their associations ("at a professional level") based on connection with technical projects, practical invention, or other technical achievements to granting it based on two more demanding requirements: (1) specific knowledge and (2) commitment to use that knowledge in certain ways (that is, according to engineering's code of ethics). The first is occupational. This requirement is now typically identified with a degree in engineering. The second is professional. Although many professions (law, especially) make a commitment to the profession's code of ethics a formal requirement for admission, engineering has not (except for licensed professional engineers, or P.E.s). Instead, the expectation of commitment reveals itself when an engineer is found to have violated the code of ethics. The defense, "I'm an engineer but I didn't promise to follow the code and therefore did nothing wrong," is never accepted. The profession answers, "You committed yourself to the code when you claimed to be an engineer."[20]

Attempts to understand software engineering as engineering have, I think, generally missed this complexity in the concept of the profession of engineering. Consider, for example, Mary Shaw's observation: "Where, then, does current software practice lie on the path to engineering? It is still in some cases craft and in some cases commercial practice. A science is beginning to contribute results, and, for isolated examples, you can argue that professional engineering is taking place.[21] Substitute "applied science" for "engineering" in the first sentence in this passage and for "professional engineering" in the second, and there is little to argue with. But, as it stands, its final sentence is simply false. There is nothing in what Shaw describes to suggest that "professional engineering is taking place."

The Fundamental Problem in Software Engineering

The term "software engineering" was coined in the mid- 1960s to describe "the need for software manufacture to be [based] on the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering.""" Thinking about software engineering thus began with the assumption that the established branches of engineering share certain theoretical foundations and practical disciplines. This is an assumption that engineers generally share, calling the theoretical foundation "science" or "engineering science" and the practical discipline "engineering method." Yet, even the history of software engineering puts that assumption in doubt.

The early proponents of software engineering disagreed concerning what engineering's theoretical foundations and practical disciplines are. Some understood engineering as essentially applied science, with a theoretical foundation in physics, chemistry, and mathematics. Others understood engineering as primarily a body of techniques for design, For them, engineering was primarily a way of moving from conception, through specification, to prototype, testing, and final fine-tuning. For most, however, engineering was primarily a way of organizing and managing a process of design, development, and manufacture, of ensuring that work would be completed on time, within budget, and to the customer's satisfaction.[23]

In fact, what the established branches of engineering share, perhaps all they share, is a common core of courses (physics, chemistry, mathematics, and so on), which may or may not provide a theoretical foundation for engineering. Beyond that, there are important overlaps between this and that field, many family resemblances and analogies, but nothing more (or, at least, nothing more of importance). For a long time, perhaps from its very beginning, engineering was a protean mix of activities held together by a common education. The common education clearly had connections with what engineers did, but the connections were not always clear, even to engineers.[24]

So, if software engineering is to be, strictly speaking, a field of engineering, it has to require of its practitioners a degree in engineering (or its equivalent).[25] Right now, the software engineering curriculum is more flexible than engineering's. It is, I think, an empirical question, one that remains open, whether students of software engineering would be better software engineers if they followed engineering's more rigid curriculum rather than, say, taking more computer science, psychology, and management courses than engineering's curriculum allows. How much physics, calculus, thermodynamics, and the like does one need to design, develop, and maintain software?

The answer to this question is not obvious. Indeed, in its present form, the question is probably unanswerable. How much physics, calculus, thermodynamics, and the like a software engineer needs may well depend on the kind of software in question (not whether or not it is "life critical" but what sort of knowledge its designer should have to do it right). Although we might worry about someone developing software for engineering applications who didn't know what engineers know, would we feel the same about such a person developing a computer game for children or a diagnostic program for physicians?

Software engineering was not born engineering. If it is ever to be part of engineering ("an engineering discipline"), it must come in by "marriage" or "adoption". That will require substantial changes in software engineering, engineering proper, or both. Software engineering may have to bring its curriculum up to standards for engineering accreditation, or engineering may have to change its curriculum to make room for software engineering (for example, by dropping the required chemistry course), or both engineering and software engineering may have to change. Software engineering cannot become engineering simply by adopting the name, by copying engineering methods, or even by having some authoritative body like the IEEE declare it engineering. Indeed, software engineers will not necessarily be members of the engineering profession even if they receive an engineering education.

Education only satisfies the occupational requirement. There is also the professional requirement, commitment to the engineers' code of ethics.[26] So far, software engineers seem to believe they can have a code of their own.[27]

Like the occupational requirement, the professional requirement leaves some room for maneuver. Software engineers can have their own code *in addition* to the engineers' code (that is, a code with obligations beyond those all engineers share). Software engineers can also try to work out a common code with engineers, changing what engineers require of themselves. What they cannot do is be engineers "at the professional level" yet refuse to share engineering's professional commitments.

Will software engineering ever join engineering's family? That is a question for prophets. What I tried to do here is to use software engineering to reveal the complexity in the concept of a profession of engineering. However, I must add that the benefits of making software engineering a "true engineering discipline" strike me as less certain than the discussion so far makes them seem. Training in engineering as such will not ensure that projects come in on time, within budget, or to the customer's satisfaction. Although engineering education has always had elements of management - more in the first half of this century than now - engineers always have problems delivering on time, within budget, and to the customer's satisfaction, especially in fields such as computer development, where experience is thin. The obvious ability of engineers in many fields to keep their promises seems more an indication of the maturity of the field than of any special knowledge of engineers as such. Aren't physicians and auditors just as able to deliver on their promises?

Nothing said here is meant to raise questions about the status of software engineering as a discipline, an occupation, or even a profession. My concern is how to conceptualize this new but already respectable occupation. Perhaps we would understand it better if we stopped trying to borrow concepts from engineering and instead borrowed them from architecture or industrial design, areas in which chemistry, physics, and mathematics are less important, pure invention more so, and codes of ethics less detailed. Or, perhaps we should borrow concepts from construction management. Software engineering may be more like overseeing the building of a great public work (a bridge, skyscraper, or power plant) than like doing the engineering for it. Construction managers are at least as good as engineers at delivering on time, within budget, and to the customer's satisfaction.[28] Or, perhaps software engineering is more like what lawyers do when they create new negotiable instruments or complex land-use agreements.

The question to be asked, then, is not whether software engineers are engineers. Clearly, while some are, most are not. The question is, rather, whether (or when) they should be.

My conclusion is that there is no fact of the matter here, only a complex of social decisions about standards of training and conduct in need of attention. Like engineering, software engineering is a social project, not a natural species.