Software Engineering: An Emerging Profession?

David Alex Lamb

September 1988
External Technical Report
ISSN-0836-0227-
88-233

Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada K7L 3N6

Version 1.2
Document prepared January 5, 1996
Copyright ©1988 David Alex Lamb

Keywords: software engineering, professions, ethics

Abstract

If we want to evolve software engineering into a "real" profession, we must not only codify a body of principles and practices, but must also develop an appropriate ethical and social framework. We must develop a clearer picture of the distinctions we want between a programmer, a software engineer, and a computer scientist. To suggest appropriate directions, this essay develops analogies between software engineering and the classic professions (doctor, lawyer, priest) as well as engineering.

Contents

1 Introduction

This essay discusses what it should mean for us to have a recognized profession of software engineering. It discusses what it means for a line of work to be a profession, and outlines my view of the body of knowledge a professional engineer should master.

These days many software practitioners have begun to refer to themselves as software engineers. However, there remains much confusion and disagreement about what it should mean to be a software engineer, how software engineering is similar to or different from programming, and whether software engineers should be licensed or certified or left to practice their profession without official interference.

In the last year I have heard several people whose opinions I respect say that software engineering is still too immature a field for us to consider it a profession. For example, at a conference in June 1988 at Rochester Institute of Technology on undergraduate software engineering education, several speakers said we aren't ready to begin designing undergraduate curricula, which are prerequisites for licensing professionals if we follow the example of engineering.[1]

However, I believe we know more about the fundamental content of software engineering than we realize. I remember discussions in the early 1970's about whether computer science was really a science. If we look back, that discipline was already mature enough, but many people remembered the old days when it seemed like a subset of electrical engineering (or mathematics, depending on who you talked to). I believe we are in a similar position with software engineering today.

My viewpoint grew out of my work on teaching undergraduate software engineering and writing a software engineering textbook [Lamb88]; Morrison and Hughes [Morr82] prompted me to start thinking about ethics and analogies with other professions. I presented some of the ideas at talks at the Institute for Retraining in Computer Science at Clarkson University, and at IBM Toronto; the audiences for those talks gave me several helpful suggestions. Margaret Lamb made several helpful suggestions on earlier drafts.

2 What is a Profession?

Unsurprisingly, we typically think of engineering as the profession on which we should base our thinking about a software engineering profession. We can improve our perspective by looking at other professions for models.

[1] Perhaps we should discuss the example of medicine instead, which requires at least two years of college before entry to medical school.

2.1 The Classic Professions

Classically (say, two hundred years ago), the professions were doctor, lawyer, and priest. Each of these professions had four key characteristics:

- Extensive schooling to master a body of specialized knowledge. This is obvious of doctors and lawyers, but perhaps not so obvious these days for priests. However, in many denominations, the priesthood requires several years of seminary; in the past, that was far more education than the general population obtained.
- A period of apprenticeship, where the budding practitioner learns to apply the specialized knowledge under the guidance of experts.
- A restricted title or license, without which one may not legally practice the profession.
- A self-governing professional organization, with the power to impose sanctions against unethical or incompetent members. Possible sanctions include revoking the license to practice the profession.

Within the last hundred years or so, engineering arose as a new profession; it followed a similar model. Most of the discussion of software engineering as a profession revolves around the body of knowledge. I believe we are close to being able to define what a professional software engineer ought to know; Section 3 discusses my summary of what is important. We need to begin to think about the other areas if we are to become a real profession.

The areas of restricted title and self-governance will give us the most trouble in establishing a profession; as evidence of their controversy, consider the years of debates about certification of programmers. To shed some light on the heated debate, it is useful to look at why professions have these two characteristics.

For the three classic professions, it was once true that the public relied on practitioners to be competent and ethical, but found it difficult to evaluate competence and, sometimes, ethics. Because of the body of specialized knowledge required to master the profession, an ordinary citizen could not tell whether decisions a practitioner made were reasonable; sometimes even judging ethical questions require specialized knowledge. Thus the public let the profession govern itself, and control who could practice the profession, in return for an implied guarantee that the professional organization would police itself. Thus each profession not only certified competence, it also enforced a code of ethics designed to protect the public.

Not all work associated with a profession needs such a guarantee. Few of the people who build bridges are civil engineers; many medical practitioners are nurses or paramedics. Such people typically work under the supervision

of a professional, although they are typically allowed to make many decisions on their own without consulting the professional. Similarly, not all software specialists need to be software engineers.

2.2 Characteristics of Professionals

From the analogies of Section 2.1, we can distinguish two key characteristics of the professional: competence, and individual responsibility.

A professional must master an extensive body of specialized knowledge, and keep up with it as it changes. The body of knowledge must consist of

- fundamental principles, so the professional has a firm foundation from which to tackle new problems and teach herself new material.
- current practice, so that the professional has something concrete with which to be sure she understands the principles, and so that she can begin to practice, at least on typical or standard problems, without having to deduce what to do from first principles all the time.

As the professional goes about her job, she not only solves the problems that are part of her work, but also tries to learn to improve her understanding of her profession.

As well as knowing how to apply principles to solve problems, the professional needs to know when *not* to act. Lawyers need to know when to refuse cases: when they are frivolous, or when the client doesn't really need a lawyer to solve his problem. Doctors need to know when not to perform surgery or prescribe medicines. Similarly, a software professional needs to know when not to program. Bentley [Bent88] has examples on the small end of the scale, such as the client who needed to generate random permutations of the numbers from 1 to 3. There are only six; the cheapest way to choose randomly among six alternatives is to roll a cubical die.

More seriously, a professional should not take on work he is not competent to perform. A solicitor who specializes in wills is not necessarily a good trial lawyer; a programmer who is a wizard at business data processing may not be very good at real-time systems. This issue is especially difficult if someone asks you to build a new kind of system no one has tried before, which happens frequently with computer-based systems. Ethically, you should propose a feasibility study or small proof-of-concept project to explore the areas of high risk; realistically, the client or your superiors may be too impatient.

An often-neglected topic in discussions of software engineering is individual responsibility. Many of the things a software engineer would do are things that many competent programmers could also do. We can imagine an intelligent, well-educated layman doing many of the things a civil engineer does, but the law requires that the plans of a major construction work have the signature of a

registered professional engineer. This important characteristic ties in with my earlier theme of protection of the public; society wants to have some assurance that its bridges won't fall down. However, you can still get your house built without consulting an engineer; some problems are well enough understood that technicians or even amateurs can handle them without a professional. Thus, we might expect future laws to require that registered software engineers certify some aspects of software systems that are important to the public, but large portions of the software business might still go to programmers rather than software engineers.

2.3 Barriers

In modern society, it may be difficult to use the classic professions as models.

- Over the years, because of human nature, professional societies and their codes of ethics gradually change to protect the profession rather than the public. In our cynical times, this brings the whole question of ethics and self-governance into disrepute.
- In the current North American legal climate, the public does not seem to be willing to allow the classic professions the same independence they had in the past. For example, patients want far more input into the decisions doctors make, and challenge apparent bad decisions in the courts with such frequency that a doctor's malpractice insurance can now cost two to three times the average citizen's salary.
- Years of title inflation have reduced the prestige of the word "profession". Indeed, Revenue Canada seems to classify any business that provides a service, as opposed to selling a product, as a profession.

However, the particular characteristic of professionals that may give the most day-to-day trouble is individual responsibility. Many employers of engineers want self-retraining technicians skilled in some disciplines technicians don't usually study, not professionals who might question management decisions on technical or ethical grounds. Reports of the Challenger disaster mentioned someone saying "take off your engineering hat and put on your management hat" – which the ethical engineer should never do.[2]

3 Central Ideas

In an engineering discipline, the specialized knowledge consists of both fundamental principles, expected to last a lifetime, and current practice, representing

[2]It might be possible to wear both hats simultaneously, which sounds rather uncomfortable and awkward.

today's implementation of those principles.

A useful distinction to draw is that a programmer might be analogous to a technician, a software engineer to an electrical engineer, and a computer scientist to a physicist. The principles distinguish an engineer from a technician; technicians usually require retraining every few years in the new practices, while engineers are supposed to be able to keep up with their field on their own.

3.1 Fundamentals

It takes a lot of time and insight to reflect on what is fundamental, so the following discussion is necessarily incomplete. However, filling out the list of fundamentals, at roughly this level of discussion, is a good approach to characterizing the field.

Clearly[3], the standard engineering activities (analysis, design, specification, implementation, validation) must be part of software engineering; different life cycle models combine them in different ways. The old waterfall model was too simplistic, but proponents of new improved life cycle models go to far in condemning it. I prefer Parnas' view that we ought to record our design decisions as though we were following a waterfall model, even though we deviate from it in many ways [Parn86].

Another key idea of software engineering is the malleability of the medium we work with. Software can be very highly flexible; this leads to strong demands for change. Moreover, software is unique in the way it blurs the distinction between components (building blocks) and processes (ways of combining components); we can capture a process in software, then treat it as a component. This gives us a way to move work out of the domain of engineers and into that of technicians, if we can capture some of our complex processes as reusable components.

Malleability leads almost directly to the need for managing complexity. We need to plan for change in almost everything we do; with most software systems we change the requirements, the system structure, the implementation details, the personnel, and almost anything else you can think of. Our main tool is the old divide-and conquer technique, separation of concerns, in several guises. Abstraction and information hiding, two sides of the same coin, are fundamental. Managing complexity requires us to document what we know, so we don't forget it, and so other people can learn it; thus software engineers must practice some form of design documentation discipline.

Each engineering discipline uses appropriate intellectual tools, often mathematical, and usually taken from the corresponding scientific discipline. Com-

[3]At first glance, it's clear. Looking more deeply, we can get mired in the discussion of alternative life cycle models. After slogging for a while, it becomes clear again.

puter science provides many useful intellectual tools, such as algorithm design and analysis, database theory, and specification methods. However, an engineer usually takes a different view of the tools than a scientist: the engineer wants to solve problems rather than codify knowledge. An illustration of a difference in attitude is that to many computer scientists, the main purpose of specifications is to prove programs correct, while for an engineer, the main purpose may be to define requirements precisely to communicate with other engineers.

3.2 Current Practice

The best of current practice isn't all that bad. We still hear plenty of horror stories, but it isn't unheard of to find shops that deliver medium-sized software projects on time and within budget with reasonable regularity. Unfortunately we suffer from very high variation in practice. Technology transfer is sufficiently slow that some important projects are using methods fifteen years or more out of date.[4]

A more subtle problem with current practice is a lack of the type of shared ground rules that might lead to consensus about what is important in the profession.

- We are especially bad at separating fundamental principles from current practice. The most common example is how we used to teach particular programming languages instead of programming principles; I believe we're getting better.
- I don't think the technician-engineer-scientist analogy is widely known or accepted.
- Few people seem to understand the necessary distinction between research journals, for communication among scientists, and professional journals, for keeping practitioners up-to-date. To illustrate the difference: I view IEEE and ACM Transactions are research journals, IEEE Software and IEEE Computer as professional journals, and Communications of the ACM as a mix, tending towards a professional journal. We have plenty of research journals, but few professional journals; there is pressure on some research journals to serve as professional journals, too (or instead).
- Within some individual disciplines, we see the emergence of "schools," each of which has its own approach, antagonistic to those of other schools.

[4]For legal reasons, and to minimize controversy, I won't give examples. You probably know of several.

> Many so-called "religious" discussions about choice of language or operating system could more accurately be called "scholastic" discussions; my academic acquaintances are often more vituperative than my religious ones.[5]

4 Where Next?

I believe the first step in developing a software engineering profession is to disseminate the viewpoint that not all programmers need to be software engineers, and that software engineering is distinct from computer science. I hope others will accept the programmer as technician, software engineer as electrical engineer, computer scientist as physicist analogy; it gives us a good outlook for drawing the necessary distinctions, and evolving each of the three fields to fill their proper roles.

Before there can be a legal framework for a profession, there must be a social and educational framework. We must begin to distinguish the kinds of work that require a programmer from those that require an engineer, avoiding the pressure to demand more credentials than any particular job really needs. We must begin to discuss the split between computer science education and software engineering education.[6] We must begin to provide the framework to help existing software specialists find their niche, bearing in mind that not everyone needs to fit in only one category.

Professional education is especially important. The Master of Software Engineering programs that some schools are developing are important; ideally, they should permit programmers to enroll part-time over several years. We need to pay a lot more attention to technology transfer; projects in this area tend to be "not well enough developed" for practitioners and "not interesting or new enough" for researchers. Finally, we need to pay more attention to professional journals, without sacrificing the research journals.

Other engineering disciplines have their handbooks, which codify existing components and practices. Mary Shaw has pointed out that as our discipline matures, we'll need to develop our own handbooks. I think developing such handbooks would both illustrate and advance the maturity of the field.

Politically, the sensible approach for developing a profession is to work with the existing engineering accreditation bodies, such as the APEO in Ontario. Such bodies are necessarily conservative, since they are supposed to protect the public and the profession rather than provide enhanced status and job

[5] I have heard rumors saying that such-and-such a journal is controlled by one school and regularly refuses contributions from other schools. I don't know whether to believe these rumors, but their mere existence is evidence of the division into schools.

[6] The enrollment in computer science programs will probably drop quite a bit after the split.

opportunities for people who want to call themselves engineers. We should work with such societies to develop accreditable undergraduate programs, and an appropriate way to license people who don't (or didn't) go through such programs.

Once this infrastructure is in place, we can begin to press for changes in the laws to require software engineers to sign off certain kinds of projects *where the protection of the public requires it.*

References

[Bent88] Jon Bentley, "Teaching the Tricks of the Trade," in *Second SEI Conference on Software Engineering Education*, Springer-Verlag (28-29 April 1988).

[Lamb88] David Alex Lamb, *Software Engineering: Planning for Change.* Prentice-Hall, Englewood Cliffs, NJ (1988).

[Morr82] Carson Morrison and Philip Hughes, *Professional Engineering Practice: Ethical Aspects.* McGraw-Hill Ryerson Ltd., Toronto (1982).

[Parn86] David L. Parnas and Paul C. Clements, "A Rational Design Process: How and Why to Fake It," *IEEE Transactions on Software Engineering* Vol. **12**(2):251-257 (February 1986).