Software Engineering: To Be Or What To Be ?

Neville Holmes
School of Computing
University of Tasmania
Launceston 7250 Australia
Email: Neville.Holmes@utas.edu.au

(Published *Software Engineering Notes*, v.24, n.3 (1999 May), 81–83.)

Abstract

This note reviews aspects of professionalism as related to computers, questions the structuring of responsibilities in software development, and commends subsuming overspecialized branches of engineering, such as Computer Systems Engineering and Software Engineering, within a more general discipline of Data Engineering.

Introduction

Some little time ago I was commiserating via email with Will Tracz over the downgrading of what had for many years been his column in IEEE Computer, *Open Channel*. Presumably because he had published several of my essays in that column, he suggested to me that I write something for ACM SIGSOFT's Software Engineering Notes (SEN) and to provoke me he airmailed me a couple of issues of SEN which turned up yesterday.

Scanning the first issue, of 1998 May, a strong feeling of dismay and déjà vu came over me. Just as, a quarter of a century ago, I had watched the Australian Computer Society rush headlong into an ill-fated push for professionalism, so it seemed that at least Texas was following their example [1]. Although the circumstances are somewhat different, the flavor of ad hockery is common to both.

This note therefore reviews the status of professionalism in fields directly related to computing, and makes somewhat belated suggestions about what kind of workers in such fields should call themselves professional engineers, compared to professional technicians.

The Computer Profession

The main arguments against a computer profession in the '70s [2] were, firstly, that no profession can base itself on specific tools, and secondly, that the computer would inevitably come into use as a tool in all professions to leave a general computer profession high and dry.

The first of these arguments can be appreciated by considering what might have happened if a telescope profession had been set up when those tools had first been developed. Had telescopes been taken under

control of a restrictive profession, their use beyond astronomy in diverse fields such as surveying and telegraphy might well have been slowed down if not blocked. Even more likely would have been the denial of use of other tools in astronomy to supplement the use of optical telescopes.

As it turned out, the widespread use of personal computers has overtaken the would-be computer profession, at least in Australia, and most education in other professions includes grounding in the use of computers.

A Software Profession ?

Even so, why not a profession of software engineering ? Surely the arguments against a computer profession don't apply to software engineering.

There are two aspects of this question—software as the basis of a profession, and software as an engineering discipline. First let us consider what the essential difference between a computer profession and a software profession might be.

If it's about computers, it's about machines. If it's about software, it's about programs. But programs run on computers, and computers are useless without programs. So where's the difference ? The obvious difference is that computers are made on assembly lines, whereas programs are made in sweat shops. On this line of argument, a computer profession would be about the use of computers, but a software profession would be about the making of software.

So far, so good. But there are two classes of software—generic and specific, general purpose and special purpose. Generic software, either operating systems or "integrated packages", can only go so far, indeed many would say they have gone too far. Operating systems are embarrassingly and needlessly complex, and lower level aspects should be transferred to hardware where they then become common to all software [3]. And since the successful general purpose packages are increasingly being provided for all popular host platforms, generic software is becoming like hardware, and is already (and sometimes controversially) bundled with the hardware. As such software throws off its feature cancer, it may become more effective as microprogram or otherwise embedded in hardware.

Thus a software profession would drift, if that's not too passive a word, to special purpose programming. Is, then, software production better done by software professionals relatively ignorant of the specialty they are supporting, or by specialists relatively ignorant of programming nitty-gritties ?

Here I am strongly reminded of experience with an automobile manufacturer who faced nearly forty years ago the project of replacing traditional DP (Data Processing) equipment with a digital computer, a replacement justified by the prospect of the computer's vastly greater application. This was before computer science graduates were available, and the company's DP management was hell bent on advertising around the world for experienced programmers to whom they would offer the attraction of (then) enormous salaries. They were with great difficulty persuaded instead to go around their existing workers, on the assembly line, in the warehouse, in the foundry, in their offices, run volunteers past a Programming Aptitude Test, and offer positions in the new programming team to the successful. The members of the team were given standard training in punched card principles, and a one or two week (I don't remember which) course in 1401 SPS programming.

The decision to use experienced workers was clearly successful. Their experience in the business was much more important for application development than their skill in programming which, because of their aptitude, was anyway quickly built up. (Another factor in this success story was that it took place before the dubious practice of separating the jobs of analysts and programmers had sprung up.)

Arguably, the situation is not all that different nowadays. At least in small and medium sized businesses, many if not most applications can be implemented with generic software, and the necessary skills in macrocoding for spreadsheet and database systems are as easy to teach and acquire as skills in assembler

programming used to be before hypercomplex operating systems became all the rage. Indeed this is happening, and the trend will grow as generic packages stabilize, Java frenzy notwithstanding.

Where does this leave a software profession? The main remaining and minority rôles are in maintenance and development of operating systems and generic software, and in the development of large and complex special purpose systems. Since large and complex systems have ever been the province of professional engineers, this brings us back to the why and wherefor of software engineering.

The introduction to (the pseudonymous?) Raccoon's encomium [4] on software engineering makes plain that the credentials of software engineering have already been rehearsed at great length in publications not available to me. Going primarily by that encomium, however, two significant aspects seem to have been overlooked in previous discussion—firstly the rôle of an engineer, and secondly the rôle of an engineering discipline.

What is an Engineer?

An engineer is the manager and facilitator of a technical process. Professional engineers do not themselves do the hackwork of their projects, though it is an important part of their apprenticeship to do some of this. The original military engineers mostly planned and supervised the work of sappers, miners, and bombardiers. Even today no civil engineers themselves drive the shovels or pour the concrete.

Engineers have craftsmen and technicians to implement their designs. Those craftsmen and technicians have their own distinct skills and knowledge, and are quite capable of undertaking small-scale projects without the help of professional engineers.

Engineers and technicians need different kinds of training. In Australia and elsewhere, engineers are produced by universities and controlled by professional institutions, while technicians are produced by technical colleges and controlled by trades unions or guilds. Because their training is longer, because their responsibility is greater, and perhaps because their numbers are fewer, professional engineers will usually earn more than technicians.

The professional engineer's skills are needed when a project has strategic aspects, when there are financial or technical risks to be overcome. The technician's skills are needed when a project has tactical aspects, when there are constructional standards and task deadlines to be met.

The striking feature of writings in the issue of SEN with Raccoon's article [4], and of other writings I peruse from time to time, is that there seems to be no division of responsibility or skills in software tasks along strategic/tactical lines. Nor is there yet any push, in this country (Australia) at least, for the systematic training of software technicians, most probably because there does not appear to be any demand for them. I suspect that this is also because of the hacker mentality—the kind of people who get into programming and succeed in it find it difficult if not impossible to move from driving computers to managing people.

However, there is an interesting hint of the need for a division of responsibilities in software development in [5] where Martin Brown attributes to B. Kitcheman (Barbara Ann Kitchenham? [5:20]) the finding of "a clear correlation between inhouse execution testing finding mainly coding errors" (tactical) "whereas customer sites mostly found design errors in released code" [5:17] (strategical).

If responsibility for software development were split between engineers and technicians, with engineers designing and managing and technicians crafting and building, then would those engineers necessarily be software engineers?

Which Engineering ?

A rather confusing footnote to Raccoon's paper [4:110] refers to Californian Assembly Bill 969 (Professional Engineers Act) under which "only civil, mechanical, and electrical engineers would be recognized" although the footnote goes on to mention also chemical engineers. The throwaway comment is that "This particular fight concerns the prestige of professional engineers, not what individual engineers do", a comment presumably not made by a Californian professional engineer, who would, one imagines, reverse the comment.

A professional engineer is an engineer first, and a disciplinarian second. Students who study engineering at a university take many of their units of study independently of their specialization, or at least they used to. When I took engineering at Melbourne University, admittedly a long time ago now, the first two years of all branches except chemical were the same, and mechanical and electrical engineering were the same for their first three years.

Some professional engineers practise in a specialty other than the one they studied, and most professional engineers move into management positions relatively early in their careers. Given the increasingly technological content of most business enterprises nowadays, it is arguable that developed nations would have been better served if they had invested in engineering education rather than ploughing so much into MBA-type courses. (This is not to denigrate such MBA courses—when I am asked, by recent graduates of the computing course I teach in, how they can best forward their careers, I usually advise them to enrol part-time in such a post-graduate course.)

So, for most professional engineers, particularly the high-fliers, the professional practice is primary, the discipline secondary. Even for the graduates of modern Computer Software Engineering courses this should be the case. Or at least so my old-fashioned engineering training would have me believe. It is therefore somewhat disappointing to find the draft accreditation criteria [6] for such CSE courses so very specialized.

But just a moment ! Doesn't CSE stand for Computer Systems Engineering, and hasn't there been a raft of that kind of CSE courses ? I'm being grossly disingenuous here, but the very real point remains that proliferating branches of computer-related engineering is a sign of immaturity, a warning to step back and think through what it is that engineers should be doing with or about computers.

Data Engineering

A quarter of a century ago, when the Institution of Engineers, Australia was organizing itself into colleges, I put forward an unsuccessful proposal for a Data Engineering College [7], defining data engineering somewhat differently than the way that term was later used. The reasoning went something like this.

Engineering disciplines are based on exploitation of a particular resource harnessed through a particular class of system. Civil engineering is based on static systems exploiting the strength and other properties of structural materials. Similarly, mechanical engineering is based on dynamic systems transforming and transmitting kinetic energy, while electrical engineering does the same for electrical energy. Data engineering therefore would be based on representational systems transforming and transmitting data.

Basic studies in data engineering would therefore encompass *data statics*—the extraction, coding, storage, and display of data; *data dynamics*—protocols and algorithms, the transmission and transformation of data; and *data mechanics*—machines for the manipulation of data. Basic studies in data engineering would be essential to education in all branches of engineering because of the increasing use of digital techniques, tools, and componentry throughout engineering. Present engineering curricula seem mainly to deal with the problem by teaching elementary programming, which is of limited value since using professional engineers to cut code would be most wasteful. Professional engineers should rather

be taught the basic principles that would fit them to manage software technicians working in projects within the engineer's discipline.

With this approach, fields like software engineering and computer systems engineering become subsumed in, or subordinate to, data engineering, or else are dealt with by technicians managed by professional engineers working in their own disciplines.

Conclusion

To anyone trained in the use of computers, and working with them full time, having a profession based squarely and fully on digital computers might well seem both suitable and natural.

However, what is important is not the computer itself, but rather the process or product the computer is used to implement or produce. The engineers and technicians who manufacture computers and programs of various kinds are continually extending the capabilities of the machinery they design or build, so the training and retraining of technicians should be continuously changing to reflect changes in the tools they use.

The education of professional engineers in basic or in advanced data engineering should be education in principles, just as it is in other branches of engineering, should be relatively independent of the tools from time to time available, should be resistant to fragmentation, and so should be much less subject to the passing whims of fashion and marketers than the training of technicians.

References

[1] Donald J. Bagert (1998): Texas Poised to License Professional Engineers in Software Engineering. ACM Software Engineering Notes, Vol.23, No.3, May, pp.8-10.

[2] W. Neville Holmes (1976): The Social Implications of the Australian Computer Society. The Australian Computer Journal, Vol.6, No.3, November, pp.124-128.

[3] W. Neville Holmes (1998): Mitigating Microsoft with Virtual Consoles. IEEE Computer, Vol.31, No.7, July, pp.105-106,109.

[4] L.B.S. Raccoon (1998): Toward a Tradition of Software Engineering. ACM Software Engineering Notes, Vol.23, No.3, May, pp.105-110.

[5] Mark Doernhoefer (ed. 1998): Surfing the Net for Software Engineering Notes. ACM Software Engineering Notes, Vol.23, No.3, May, pp.13-19.

[6] Gerald L. Engel and Richard J. LeBlanc (1998): Draft Accreditation Criteria for Software Engineering. ACM Software Engineering Notes, Vol.23, No.3, May, pp.11-12.

[7] W. Neville Holmes (1974): Data Engineering. In the Proceedings of the Conference on Computers in Engineering, The Institution of Engineers, Australia, Sydney, May 16-17, 1974, pp.111-115.