A Manifesto for Software Engineering

H. James Hoover
Software Engineering Research Lab
Dept. of Computing Science
University of Alberta
hoover@cs.ualberta.ca

> *Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints. Architecture must include engineering considerations, so that design will be economical and feasible; but the emphasis in architecture is upon the needs of the user, whereas in engineering the emphasis is upon the needs of the fabricator.*
>
> Fred P. Brooks, Jr.,
> Architectural Philosophy,
> W. Buchholz, Ed. Planning a Computer System. McGraw-Hill, 1962
>
> *To be a programmer is to develop a carefully managed relationship with error. There's no getting around it. You either make your accommodations with failure, or the work will become intolerable.*
>
> Ellen Ullman,
> The Myth of Order,
> Wired, April 1999

What is a profession?

The notion of a regulated profession arises when the practices of some recognized group have the potential for significantly affecting the public good. In particular, when the malpractice of the group has potential for causing significant harm. So, when considering whether an activity should become a regulated profession one must ask these questions:

- Does the activity have the potential for significant adverse impact on public welfare?
- Is there a well-defined notion of practice and malpractice?
- Is there a recognizable group of individuals performing the activity?

Without question, the activity of running programs on computers has enormous impact on society, both good and bad. As for the other two questions it is not clear that they have affirmative answers. However, let's suppose that there is a potential for a profession of "software engineering".

So what is software engineering? Part 1.

Who knows? In fact who cares? The breadth of application of software is so huge that assigning the task of responsible design and construction of it to any one profession is meaningless. One might as well talk about mathematics engineers, or organizational engineers.

Why do I say this? Software is just the latest reflection of a history of humans inventing and using processes. In fact, I claim the whole purpose of software is to implement process. The fact that software is involved in the process is a distraction, the real issue is the role and quality of the process being implemented.

Yes, there is a scale difference in that computers do more, do it faster, and so we can perform more complex processes. But this complexity is nothing new either - complex processes have been implemented in large bureaucracies for a period far longer than either engineering or computing science have existed.

Do engineers have a special claim on process?

Engineers design and build artifacts: static, dynamic, and processes. But architects design and construct buildings, chartered accountants and lawyers design and build company business processes, physicians design and implement processes for combating disease. Is an engineer that writes software for an embedded application a software engineer? Is a physician who designs experiments and analyses them on a statistical package a software physician? Is an accountant who builds the corporate financial model a software CA?

An appropriate analogy can be made with mathematics. Mathematics, to a greater or lesser degree, is used in each of the previous examples. No one would seriously propose that only qualified government regulated mathematicians should have done the mathematics. Instead, the proper application of a tool is deemed to be the responsibility of the profession using the tool. It is the responsibility of the engineer, architect, physician, accountant, or other professional who uses the mathematics to know when they are competent, and when they should call in an expert for advice.

Software falls into the same category. We want every profession to have appropriate knowledge of computing science. And in particular, we want them to know when they are exceeding their expertise. The accountant has to know when to call in the data modelling and workflow experts, the computer engineer has to know that you don't build safety critical systems if you don't understand the notion of a critical section.

So what is software engineering? Part 2.

If software engineering is the discipline of building software, then there is no question that computing science has staked its claim. Computer scientists, quite simply, build more software than any other group. Despite our failures, (i.e the computer science equivalents of Three Mile Island, Chernoybol, Bohpal, Challenger, DC 10 cargo doors,

Titanic, and so on.) we know more about building software than any other group. Some of us are quite good at it, and constantly getting better. Perhaps in another decade we will even have well defined enforceable notions of best practices and malpractice.

But educating a computer scientist is as lengthly and difficult as educating an engineer. You can just barely do this in 4 years. Anyone who thinks that you can be both trained in computer science and another engineering discipline in one undergraduate program clearly has no grasp of either field.

Frankly, I do not think even a computing science B.Sc. grad is ready to be called a software engineer. They may know a lot of computing science, but they have little experience of how to apply that knowledge to serve the needs of clients as professionals. This is exactly like newly minted engineers, accountants, physicians and so on. If anything, software engineering should be a master's level discipline that requires substantial practical experience.

I am going to adopt the following working definition:

Software engineering is the broad activity of the design and implementation of processes that are intended to be carried out with the assistance of computers.

Software engineering belongs to no one discipline. It is common to all disciplines that design and implement processes. Its "software" roots clearly rest in computing science. But its notions of professional practice come from engineering. It affects the way organizations work, and whether they succeed or fail, and so it has aspects of business to it. Software affects people, and so it has aspects of psychology, anthropology, and sociology. It affects our rights, and so has aspects of law and political science. I claim it is the epitome of interdisciplinary subjects, but its home base is and always will be in computing science.

Turf Wars and Software Engineering

Frankly, I am fed up with the turf wars over Software Engineering. The engineering profession dropped the ball 20 years ago when I was an undergraduate in Computing Science. I was planning on entering Electrical Engineering but realized that most engineers thought of computing as "just programming". No EE program could provide me with the combination of theoretical foundations and practical insight that CS did at the time, and I claim none can do it now.

Software engineering is a discipline that began in the late 1960's. It was started primarily by computer scientists, some mathematicians, a few physicists and some members of the software industry. The founding meeting (NATO Workshop on Software Engineering - 1968) had no academics from engineering schools. The development of the discipline over the past 30+ years has been primarily from the results of researchers in computer science departments. In the past 5 years the discipline has really exploded and there are now many of computer engineering faculty and business faculty that are also conducting

research in this area. Their participation and efforts have helped define the problems and the area, yet the bulk of the research concentration continues to be in computer science department or in computer science and engineering department that are common in the US where the engineering professional has taken a less exclusive stance with respect to engineering programs.

The engineering curriculum as it stands cannot possibly produce competent software engineers with serious computing science depth and still maintain the goal of a shared experience with other branches of engineering. To make room for the requisite computing science requires dropping so much of the traditional engineering curriculum that it would no longer look like an engineering program but more like a computing science program. Such a "software engineer" would have little in common with a traditional engineer, except perhaps the tribal mentality deliberately spawned by Herbert E. T. Haltain and made poetic by Rudyard Kipling.

The executive summary of my position is this: Software engineering is a computing science discipline, we got there first, over 25 years ago. If computing scientists are not allowed to use the term "engineering", than engineers are not permitted to use the term "software".

But I would prefer that APEGGA (Association of Professional Engineers, Geologists and Geophysicists of Alberta, Alberta Province in Canada) accepts that engineering disciplines change over time, that new ones arise that are outside its mandated scope, and that rather than attempt to use a government sanctioned monopoly to make up for past short-sightedness, it should embrace software engineering as multi-disciplinary, and expand APEGGA to include computing scientists. Such flexibility should be possible.

I contend that parochial concerns about who should and should not use the term "software engineer" are diverting us from our proper obligation to ensure that processes are designed and implemented properly.

Instead, I think we have an opportunity to build a unique and relevant kind of software engineering on campus. It is not just a combination of computing science and computer engineering. It should at least include the MIS aspects of the faculty of business. It is more, but I think we should focus on just these three.

Why? Business and governments are the biggest producer and consumer of software as part of their business processes. Most software disasters, both in implementation or use are in business process software. So if we want to have an impact on society, this is a good place to invest our efforts.

The challenge is this: Do we fight over turf, or do we build a discipline that makes us recognized as being *the* place that understands how to design and build software that serves the needs and solves the problems of clients and society.