Is Software Engineering Really Engineering?

Lawrence Peters, Software Project Management Consultant

Abstract. In an attempt to improve the cost and quality issues in software, more than 40 years ago, NATO sponsored a conference which may have formally introduced the term, "Software Engineering" to the world. The term held out hope that the software community would improve matters if they viewed what they were doing as engineering rather than some form of cottage industry based on computer code. The fact was that what engineering meant as a profession was unfamiliar to most software developers at that time and still is a serious challenge to today's software professionals.

At the time of the first NATO conference [1] on software engineering, people from a broad range of backgrounds had found a home in software development. So much so that studies were done to find out just what it was about software development that drew in people from such a diverse set of backgrounds [2]. What they found was a predominant psychological profile consisting of a pair of attributes that were unique when taken together. Of the 60 professions studied, software developers were unique in that they had:

- **High GNS (Growth Need Strength)** – This represents a strong desire to solve significant, difficult problems, particularly problems which were at the leading edge of technology or moved the technology frontier forward. If you have ever wondered why it is so difficult to get software engineers to document their work, it could be due to this trait.
- **Low SNS (Social Need Strength –** This represents a strong desire to work alone rather than as a contributing member of a team. Most upgrades to existing software systems are complex enough that a team of people are required to accomplish them. This trait makes building that team more difficult than one might expect.

These characteristics were present regardless of the discipline these folks were originally trained in. For example, the top 10 software developers I have ever worked with over the last 40 years include a high school dropout, an M.S. in Psychology, an M.S. in Library Science, an Art History major, a mathematician, a theoretical nuclear physicist and others (including a couple of people with engineering degrees) from a total of 5 different countries. Granted, this is only one person's experience but those who have been in the software field for more than 20 years probably have had a similar experience. At some time during their academic or professional careers, people with this profile get exposed to computer programming and become "hooked" abandoning their original vision of how they would make their living in this world for what we now view as software engineering. Another attractant of software engineering is the fact that the work one does can be highly personalized, expressing one's own "style" and not having to conform to some rigid rules written by others. Oddly enough, this is in spite of the fact that the syntax and semantics of the programming language one uses are not open for discussion but they set the bounds within which a program can be created.

Today, the population in software development appears to be much more uniform perhaps due to the popularity of college majors like computer science, information systems and others including, software engineering. At this point it is fair to ask, "What is engineering?" The international organization tasked with defining and maintaining definition of the term engineering (ABET Accreditation Board for Engineering and Technology) defines engineering as, "the profession in which knowledge of the mathematical and natural sciences gained by study, experience, and practice is applied with judgment to develop ways to utilize economically the materials and forces of nature for the benefit of mankind." Under that definition, software engineering is a "rather odd bird." In the more traditional branches of engineering, we are dealing with the forces of nature (e.g. physics, chemistry) which continue to work as they always have or at least, we think so. Software is definitely different in that nearly everything we deal with during even relatively simple projects changes – the requirements, budget, schedule, user interface all undergo nearly continuous evolution over the course of the project.

At this point, we can conclude that software engineering is not really engineering at all or it is the prototype of a new form of engineering dealing in the realm of logic as it is applied to functionality to benefit mankind. Let's take a look at each of these viewpoints:

- **Software engineering is not engineering –** A less restrictive definition of engineering is, "the process of developing a system that conforms to accepted engineering and development practices meeting or exceeding functional, performance and reliability requirements." Under this one, software engineering qualifies as engineering provided the term system includes software.
- **Software engineering is engineering** – Under the previous definition, software engineering survives classification as engineering. But it also has adopted many of the accoutrements of the more traditional engineering disciplines. These include the development and refinement of a body of knowledge (SWEBOK) [3], standards for various activities comprising the software development process and conferences and research in various parts of the world focused on specific topics of concern to software professionals.

A concern shared by many software engineering professionals is the seeming inability of our profession to meet budget, schedule and quality issues. This appears to be a symptom of our parochialism because other, well-established branches of engineering miss their targets with a great deal of consistency [4].

Figure 1:

Type of Project World Wide	Average % Overrun	Overrun % Range
Railways	45	7 to 83
Bridges & Tunnels	34	-28 to 96
Roads	20	-10 to 50

It turns out our seeming inability to estimate with a fair amount of accuracy is actually a characteristic shared by all human beings [5]. Identifying it resulted in a Nobel prize in Economics in 2002. He demonstrated that everyone misses estimates due to 2 factors:

1. An over estimation of our abilities – Just about every software project manager has sat in a meeting with the development team to discuss the cost and schedule issues related to a proposed change and have the team indicate the change would be a "piece of cake" only to find that when the change is actually put into place it is a cost and schedule nightmare.

2. An over estimation of the benefits of the effort to the near exclusion of consideration of risks – In combination with #1, this factor is particularly dangerous when submitting a bid for a contract in competition with other firms. The team looks at how much they can make on the contract, play down risks that can cause overruns and ignore the complexities inherent in the proposed effort.

Since these are traits common to us all, no branch of any profession has mastered accurately predicting the future. However, there is a way to back out this over optimism [6] changing the estimating problem. It involves a process whose conclusion results in the cost estimate being tempered by an x-y graph of dollar amount versus confidence level. The dollar amount represents the size of the set aside (also referred to as a "contingency fund"). The higher the dollar amount, the higher the confidence level that the project will be completed within the budget + contingency amount. The slope of the line used to identify the contingency fund level. The data used to compute this chart is based on your team's experience in estimating and executing similar projects. It is so effective that the American Planning Association has recommended its use over traditional planning and estimating methods [4].

But the key issues in all of this remain:

- What is engineering?
- Does software engineering qualify as engineering?

Dictionary definitions of engineering provide a relatively straightforward benchmark for us. According to Webster's dictionary [7] engineering is defined as: "The design and manufacture of complex products"

The Problem

No matter what facet of software engineering people work in, they are reluctant to identify themselves as a, "Software Engineer." This may be due to some social image of engineers as being boring or old fashioned or something else unfashionable Perhaps this will change as the public's awareness that software engineers make a decent living and work in a generally low hazard environment. A part of the problem we have not addressed is the issue of ideology or belief system. What people belief or accept as fact is extremely difficult to change. Dissuading a belief involves a lot more than facts and data [8]. A crisis or some other catastrophic event may accompany a change but this is not always certain [8]. It all depends how we attribute the failure or misinformation – to ourselves or some other factor [9].

Regardless of what you define as engineering, all engineers work in an environment marked by:

- There is a problem to be solved
- Solving the problem is complex because it entails solving many subsidiary problems
- The solution will benefit one or more companies, governments, or members of society
- Arriving at the problem solution must be bounded by the budget and timeframe allotted
- There is no guarantee that the problem can be solved

Also, engineers generally follow a common process which generally proceeds as follows:

1. There is a problem to be solved
2. The problem and its proposed solution are documented in a way that both the client and the engineering team can understand
3. A design is proposed and cost estimate developed
4. If approved by the client, implementation of the design proceeds via a project plan, costs detailed and prototypes and/or mockups developed together with acceptance criteria and so forth.
5. Upon completion and compliance with the acceptance criteria, the system is delivered

As we can see from this over simplified depiction of the engineering process, all engineering processes share strong similarities. The IEEE has made the some similarities even stronger via the creation and maintenance of a series of standards and the Software Engineering Body of Knowledge [3]. In addition, the Software Engineering Institute developed the Capability Maturity Model and its successors consistent with ISO 15504 and ISO's efforts worldwide to assist organizations improve the quality of engineered products.

What Can We do to Promote the Use of the Term Software Engineer?

If you take a look at the schools that offer software engineering as a major at the graduate or undergraduate level you will find these programs are run under the auspices of the computer science department, the information systems department and other department titles – not the school of engineering. Why? I am not sure but more than 30 years ago, in authoring the first graduate level curriculum in software engineering at Seattle University, we made sure it was part of the school of engineering. Students entering that program knew right from the start that this was an engineering discipline and, if successful in it, their degree would be awarded by the engineering department. What has happened since then in more than 100 universities around the world is the creation of software engineering programs mostly run by the computer science department [10]. An examination of these curricula reveals that the adage, "We teach what we know" still holds true. The programs are largely about programming methods, general condemnation of the Waterfall lifecycle and often sprinkled with esoteric topics related to code production. Even after all these years, the United States Bureau of Labor Statistics in its latest update [11] online lists many categories of engineer and engineering but does not list software engineer as a labor category. Perhaps it is a lack of self-promotion or the comparative newness of this discipline which has prevented software engineering from being recognized as a legitimate branch of engineering. For example, civil engineering has been around since before the time of the ancient Romans.

Is There an Alternative?

If we did not call this profession Software Engineering, what would we call it? Alternatives do not readily come to mind especially since this label has been around for so long. Perhaps the best course of action is for us to gain more knowledge of other engineering fields and to identify ourselves as engineers. This may go a long way to making Software Engineering a permanent part of the engineering lexicon.

Disclaimer:

ABOUT THE AUTHOR

Dr. Peters has four decades experience in software engineering as a software engineer, project manager, consultant, and educator (he currently teaches Software Project Management in Spain via Skype). He has worked on many defense systems. His clients have included many Fortune 100 companies, the US Department of Defense, and the Canadian Defence Establishment. He has a B.S. in Physics, an M.S. in Engineering and a PhD in Engineering Management. He created the first Software Engineering laboratory for the Canadian Air Force, written 4 books and published several papers

E-mail: ljpeters42@gmail.com

REFERENCES

1. Naur, P. and Randell, B. (Editors), "Software Engineering: Report on a Conference Sponsored by the NATO Science Committee," Garmisch, Germany, 7th to 11th October, 1968, Brussels, Scientific Affairs Division, NATO, January, 1969.
2. Couger, D. and Zawacki, R., "Motivating and Managing Computer Personnel," Wiley-Interscience, New York, N.Y., 1980
3. IEEE, "Guide to the Software Engineering Body of Knowledge," Version 3.0, available for download, December, 2014.
4. Flyvberg, B., "From Nobel Prize to Project Management: Getting Risks Right," Project Management Journal, August, 2006, pp. 5 – 15
5. Kahneman, D., 1994, New challenges to the rationality assumption. Journal of Institutional and Theoretical Economics, 150, pp 18-36
6. Peters, L. J., "Managing Software Projects: On the Edge of Chaos – From Antipatterns to Success," Kindle ebook, Software Consultants International Limited, 2015.
7. Merriam-Webster on line dictionary, December, 2014.
8. Festinger, L., Riecken, H. and Schacter, S., "When Prophecy Fails," Martino Fine Books, Eastford, Connecticut, 2012
9. Myers, C., Staats, B. and Gino, F., "'My Bad!' How Internal Attribution and Ambiguity of Responsibility Affect Learning from Failure," Harvard Business School Working Paper 14-104, April, 2014.
10. Peters, L. J., Unpublished Preliminary Results of Internet Search in Support of Software Project Management Course as part of EMSE Program, Universidad Politecnica de Madrid, 2014.
11. United States Department of Labor website job categories listing, latest update, January 8, 2014.