Borrow Fundamentals from Other Science and Engineering Disciplines

Franck Xia
Department of Computer Science
Saint John's University
Collegeville, MN 56321
320-363.2837 / fxia@cs.csbsju.edu

Abstract
This paper articulates some fundamental requirements of science and engineering widely observed in traditional engineering disciplines and argues that we need to adopt in SE research a similar methodological rigor. SE community needs to define rigorously technical concepts, formulate research hypotheses, test theories, and reexamine conflicting theories closely to maintain our theory consistent.

Keyword: Engineering, methodology, theory, definition, hypothesis, hypothesis testing

1. Introduction
In this paper, we will elucidate the common ground of science and engineering and analyze the lack of theoretic foundation in software engineering in comparison with other well-established engineering disciplines, and we will argue that we need to borrow fundamentals from other disciplines and change research methodology in software engineering.

Nobody will doubt that there is a significant difference between science and engineering or between software engineering and other well established engineering disciplines. Science explains the nature while engineering focuses on production. Traditional engineering deals with materials while software engineering investigates how to manipulate information by human being. Despite of their differences, however, there is also no doubt that they have some common features. This paper will articulate some common ground of science and engineering.

We can reasonably assume that we are not interested in comparing software engineering with other medieval engineering arts based on some empirical rules derived very much from trial and error experiments. In fact, the history of mechanical engineering can even be traced far back in antiquity, e.g. the construction of Pyramids in Egypt [1].

When we compare software engineering with optical engineering, electrical engineering, and mechanical engineering in their modern sense [1-5], we find there is a strong contrast and notice that software engineering shares few common ground with others to such an extend that we should question what is our comprehension of engineering.

2. What is Engineering?
This problem seems so trivial that, as software engineers or researchers, we do not even want to bother ourselves with this sort of "pedantic" question, for we are software engineers and people would say that we should know the answer. Yet this is not always true and not all engineers or researchers could articulate clearly what are the basic requirements of engineering disciplines, just as not all individuals are able to explain what is the definition of intelligence, though we certainly use our intelligence for problem-solving everyday.

2.1 Definition of Engineering
Engineering is circumscribed, in Webster dictionary, as the *science* by which the properties of matter and the sources of energy in nature are made useful to man in structures, machines, and products [6]. Oxford dictionary circumscribes engineering as a field of study, especially that part of it which can be treated according to the *laws of mathematics and physical sciences* [7].

By the above two dictionary definitions, it is clear that 1) engineering must have scientific theory as foundation, and 2) the application of theory to develop methods or tools is an implications for building engineering products. Nevertheless, people may still question whether or not these definitions reflect the engineering reality. Let us ascertain first that mature and well-established engineering disciplines do have scientific theory as foundation. And this should be the common ground of science and engineering.

The theoretic investigation of Newton and Huygens on Optics proceeded far more before any engineering practice. Without Snell's law, imaging theory for ideal optical systems, Gauss optics dealing with lens systems as an approximation of ideal systems, and geometrical aberration theory for control the quality of imaging systems, there would be no classical optical engineering. Modern optics studies the formation of images, interference, diffraction, etc. based on the theory of electromagnetism [3].

Electronic and electrical engineering are firmly founded on Ohm's law, Kirchhoff's law, Thevenin theorem, Norton theorem, Faraday's law, Lenz's law, etc. Many theorems or laws can find their root in the theory of electromagnetism through Maxwell's equations [2,4,5]. Note that it is impossible to imagine how electric motor, which triggered further the Industrial Revolution, could be invented without the theoretic discovery of Ampere and Faraday on the relationship between electricity and magnetism.

Civil engineering relies on classical mechanics, built since Newton, mechanics of materials in general including the basic Hooke's law and elasticity theory, mechanics of soil and concrete, in particular, and hydraulics [1,4,5]. Historically, the theory of mechanics developed in 17th and 18th centuries laid down the groundwork for the Industrial Revolution in 19th century; with the advent of stream engines, laws of thermodynamics were discovered [4].

2.2 What is Software Engineering?

Software engineering has evolved in a particular way, for and the industrial development of software preceded our theoretical research. This contrasts significantly with other mature engineering disciplines such as optical, electronic, electrical, and civil engineering for which theories have been established and tested first. For this reason, software research is largely driven by business needs for providing practical solutions and researchers tend to neglect the necessity of theory for engineering. The IEEE standard reflects this narrow view which defines software engineering as a) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and b) the study of this approach [8,9].

However, this definition is flawed. First, craft may also be systematic, disciplined as well as quantifiable. Countless examples of systematic, disciplined craft production can be found in the early days of human civilization [1]. Secondly, study does not imply the development of theory. For instance, case studies are normally limited to describe practice and draw some empirical lessons rather than developing theory. As an extreme case, numerology requires also studies that involve mathematics.

Methods and tools are also used in craft for building systematically products. From the dawn of civilization, people knew how to build, with different *methods or tools,* grandiose edifice with wood or stone such as Notre-Dame de Paris, or to manufacture magnificent furniture. As academic, we teach at college with pedagogical methods or tools extensively. Still we do consider ourselves as engineers despite that education could be regarded in certain sense as a mass production. We are simply artists because there exists no theory for our craft activities.

Therefore, we must understand that *methods, tools, or scale of production merely determine the maturity of craft or engineering, but do not distinguish them; what differs engineering with craft first and foremost is its theoretic foundation.*

Due to this too pragmatic but erroneous view of software engineering, we have developed a great amount of useful techniques, methods, and tools in software engineering, but the basis of engineering - theory - seems to be missing. Yet, a real engineering discipline, mature or not, needs theory. The above analysis demonstrates that this seemingly trivial question of what is software engineering is by far pedantic. When we fail to provide a correct answer, there are serious methodological consequences.

3. Some Theoretic Requirements of Science & Engineering

One may argue that software engineering has already so many theories developed in the past decades. Thus the above discussion sounds meaningless. The following discussion, though not exhaustive, might help researchers in SE to clarify a fundamental issue which seems to be well understood in traditional engineering: what is scientific theory?

3.1 Technical Terms Must Be Rigorously Defined

No theory can be elaborated based on vague and undefined concepts. With vague terms, there is no logic basis for any scientific discussions. Even in most western legal systems, basic terms must be

unambiguously defined so that the application of laws can be as rigorous as possible to avoid arbitrariness. Thus technical terms must be defined unambiguously in engineering. We cannot imagine what mechanics will look like if, for instance, velocity, acceleration, energy, or strength have no clear definition; or current, capacity, and resistance have no rigorous meaning in electrical engineering; or reflection, diffraction, and aberration have no definition in optical engineering. Note that concept formation/definition has always be taken seriously in science, because we know since Aristotle that this is the beginning of scientific discourse [10].

However, this trivial and elementary requirement of science has not been fully realized in SE. The erroneous definition on the most basic concept of Software Engineering aside, some fundamental and unanimously accepted concepts, e.g. coupling and cohesion, have been defined ambiguously [11]. Other concepts such as size, length, complexity, and quality have no clear definition at all [12]. Although we know roughly what analysis, design, coding, and testing are about, these important concepts still lack an unambiguous technical definition, especially for analysis or design.

One of the direct consequences of the lack of definition is that we can hardly apply mathematics. That is the reason why we have so many informal methodologies. Formal methods are an exception which focuses on correctness of software and brings rigor. Yet, as many important quality concepts such as complexity have no definition, they cannot be treated in a formal way. Another consequence is that when we apply process control model, originally developed in manufacturing industry, one important aspect relating to concept definition is overlooked: in traditional manufacturing industry, people know how to measure the quality of products whereas in software engineering we even do not know how to define these quality concepts, let alone their measurement. It is true that bureaucracy can avoid anarchy, but it is hard to justify that by emphasizing on process control, we will be mature and can guarantee product quality knowing that we do not even integrate the measurement of these quality attributes in the process. This is the case of CMM that based its measurement heavily on defect ratio to control product quality [13]. We may wonder, for example, how this magic ratio could ensure maintainability which is an important aspect that most engineers should consider.

If we are interested in borrowing ideas from mature engineering, the first one we would suggest to borrow is to try to understand software technical concepts and define them unambiguously.

3.2 Engineering Theory Must Have Hypothesis

But even terms in a discourse are well defined, there is no guarantee that the discourse will contain theory. Theory aims to explain things. When we explain, we need to justify our explanation, or in other words, answer the answer “why” [14,15]. In general, our justification could be either determinist or statistic. When reasoning is determinist, we use cause-effect relation to explain things. When reasoning is statistic, we resort to statistic theory to explain things. In both cases, the reasoning is based on certain verified hypotheses. Formulating hypotheses is one of the most difficult and important tasks in empirical science and engineering. Yet, in SE research, the role of hypothesis has not been taken seriously. Oftentimes, methods were proposed with *vague or exaggerated hypotheses*. For instance, instead of saying “F.M. can improve software correctness", people claim “F.M. can improve software quality” which is extremely vague due to the lack of definition on quality and consequently becomes finally overly exaggerated, for quality is generally understood as a multi-facet concept consisting of many attributes and correctness is just one of them. Other quality attributes can hardly be treated by formal methods. When people suggest that DFD, Data Dictionary, and Process Specification using Decision Tree or Decision Table are good or efficient tools for developing software specification, we don’t know what “goodness” or "efficiency" means. In other words, there is no technical hypothesis about the usefulness of these specification tools, and hence there is no logical ground to predict the result of using these tools.

When hypotheses exists, generally the merits of SE methods are surprisingly *unconditional*. The absence of condition for the merits of F.M. implies that they do not depend on other factors, e.g. time constraint; claiming the merits of systematic reuse has the same problem which overlooks the very specific nature of software and software production.

3.3 Theory Must Be Consistent

Scientists are reluctant to accept inconsistent theories. In a traditional scientific discipline, when a new observation or law contradicts other the existing theory, the whole theory must be suspected and a new theory is searched to replace the existing one. It might well be that some conflicting theories could explain different kinds of phenomena. But

scientists are extremely cautious before they come to accept contradictory theories. They want to make sure that the nature can only be explained by different and contradictory theories but not one theory [15]. Quantum Mechanics is such an example which contradicts classical Mechanics.

In SE, it seems that people do not feel uncomfortable with contradictory ideas, principles, or theories. Here are some examples of contradictory ideas or principles that the author find particularly amazing: In reengineering, people agree that we have to consider cognitive aspects of software comprehension, whereas in software measurement many researchers refuse to consider comprehension aspects of software; we all believe the principle of abstraction, but Formal Methods adhere to the principle of rigor which is implicitly against abstraction because abstraction is based on essence which must neglect details and leave space for ambiguity; the principle of inheritance in OOP appears to be in conflict with the principle of reducing coupling, yet some proponents of OOM called it a healthy contention [16]. This seems to be unique in engineering, because we cannot find other engineering theories based on conflicting principles.

It is worthnoting that conflicting requirements of software development cannot justify conflicting theories, just as the existence of evil things in this world cannot justify the existence of an evil theory. We may have competing engineering theories, but not theories based on contradiction. Theory must rely on logic and we know since Aristotle that the law of excluding the middle excludes contradiction in theory.

3.4 Theory Must Be Tested

Cause-effect relation revealed in hypotheses must be validated through controlled experiments either deductively or statistically. Testing must not rely on subjective evaluation or intuition but only on quantitative measurement. We must also understand that correlation cannot prove causation. In Physics, people may spend years or even decades to test a theory. It is worthy to note that this basic requirement of scientific inquiry has been well understood now by researchers in social science, behavior science, or even business research. Unfortunately, there is an obvious absence of hypothesis testing in SE. Our "theories" are based on lessons leant, beliefs of experts, or what the others said. We call them theory, not because they have been subjected to serious testing but because they seem to be useful. This is another misunderstanding of scientific and engineering theory. Until now, there exists no controlled experiment to compare Formal Methods with others; few controlled experiments to compare OO with Non OO methodology can be found in our literature. Fortunately, some comparative studies on OOM surfaced recently which brought about a controversy [17]. To some extent, this lack of hypothesis testing is understandable, because we are not bothered with formulating hypotheses for our theory and hence there is no hypothesis or theory to test!

Maybe we can understand better the methodological difficulty in SE research through a recent attempt of IEEE Computer Society trying to determine software engineering principles via a survey, or in other words opinion poll. The intention of the survey is to build a theoretic ground for software engineering. However, this idea is bad not because it has never been used in other established engineering but because even in politics, social science or business research nobody resorts to poll for selecting principles! We accept principles because based on them we can explain things deductively. Building principles based on opinion poll, we will definitely lose the objective ground of engineering.

4. What We Really Need to Borrow?

It is true that engineering has a practical purpose and does not necessary need rigorous theories. However, theories are still needed. For instance, Gauss optics is an approximate theory which is accurate enough and can be tested. In software engineering, we can find few theory of this kind. One may argue that software engineering is really special because what we are dealing with is not material but information. This argument is untenable because we have in computer science a pretty rigorous way to define constructs of programming languages, semantics, complexity of algorithms, etc. One may also argue that software engineering involves human being intelligence which is hard to formulate. Nevertheless, we believe that few researchers in cognitive science would agree that they cannot conduct scientific investigation to develop theory. So we fail to see why we cannot develop theory in software engineering.

If we wish to build a sound SE discipline and if we think it possible, more attention should be dedicated to its theoretical and methodological aspects which are routine practice in traditional engineering. From the above analysis, we reach the following conclusions: We should realize that basic concepts

must be clarified and defined first, theories advanced with nebulous terms must be suspected systematically until the terms are clarified; that hypotheses must be explicit and conceived carefully, any methods should not be proposed with gratuitous labels such as "good" or "effective" but with a technically testable hypothesis such as "reducing maintenance effort"; that conflicting principles or hypotheses cannot be accepted and must be reexamined closely; and that theory must be tested with elaborate experiments. With this science and engineering rigor proved by the history, we can certainly expect to build a sound software engineering theory to better guide software practice.

Reference

1. Burstall A.F., A History of Mechanical Engineering, The MIT Press, 1965
2. Kinsford P.W., Electrical Engineering - A History of the Men and the Ideas, St. Martin's Press, N.Y., 1969
3. Born M. and Wolf E., Principles of Optics, Electromagnetic Theory of Propagation, Interference, and Diffraction of Light, 6th ed., Pergamon Press, 1986
4. Garrison E., A History of Engineering and Technology, CRC Press, 1991
5. Gillmor C.S., Coulomb and the Evolution of Physics and Engineering in Eighteenth-Century France, Princeton Univ. Press, 1971
6. Webster's 3rd New International Dictionary, Merriam-Webster Inc., 1986
7. Oxford English Dictionary, 2nd ed., Clarendon Press, Oxford, 1989
8. IEEE Standards Collection: Software Engineering, IEEE Standard 610.12-1990, IEEE 1993
9. Pressman R.S., Software Engineering: A Practitioner's Approach, 4th ed. McGraw-Hill, 1998
10. Hempel C.G., Fundamentals of Concept Formation in Empirical Science, Univ. of Chicago Press, 1972
11. Xia F., On the concept of coupling, its modeling and measurement, J.S.S., vol. 50, Jan. 2000, 75-84
12. Xia F., Look before you leap: on some fundamental issues in software engineering research, Information & Software Technology, vol. 41, no. 10, 1999, 661-672
13. Humphrey W.S., Managing the Software Process, Addison-Wesley Pub., 1990
14. Fowler W.S., The Development of Scientific Method, Pergamon Press, 1962
15. Dampier W.C., A History of Science, Cambridge Univ. Press, 1979
16. Booch G., Object-Oriented Analysis and Design, 2nd ed., Benjamin Cummings, 1994
17. Hatton L., Does OO Sync with How We Think ? IEEE Software, May/June 1998, 46-54