

Teaching Software Engineering: Problems and Suggestions

Osama Shata

Department of Computer Science and Engineering, Qatar University
Doha, Qatar

Abstract

Teaching Software Engineering is a challenging task. This paper presents some problems encountered during teaching the course of software engineering to computer science and computer engineering students for few offerings. We present problems encountered and which are related to its title and contents and present suggested solutions.

Keywords: *Software Engineering, Development Cycle, Object-Oriented.*

1. Introduction

This paper presents some problems encountered during teaching the course of software engineering to computer science and computer engineering students for few offerings. We present problems encountered as well as suggested solutions.

I teach Software Engineering, which is a common compulsory course in many Computer Science and Computer Engineering curriculums. Probably because programming courses are part of those curriculums and software engineering is being defined, in general, as concerned with developing quality software. However, I found that in many cases, and during my discussions with colleagues on how to improve the course, the course is being looked at as an intruder to both curriculums. Computer scientists do not feel that it is a real computer science course. While the word "engineering" in its title may be contributing to this feeling, the course is also different in its nature from real computer science courses such as Computer Architecture, Operating Systems, Algorithms … etc. Computer engineers also do not feel that it is a real Computer Engineering course, probably the word "software" is contributing to this feeling, but more importantly, it lacks hardware components and lacks real design experiences. This paper begins with a brief introduction to the origin of the Software Engineering discipline. Next the paper will discuss contents that are usually being taught in a typical Software Engineering course and highlights problems faced and offer suggestions. The paper concludes with a summary.

2. Problems

The term Software Engineering (SE) was first introduced in 1968 in a NATO conference to address software crisis which came to surface in that period, when many large software projects faced great difficulties such as unexpected delay in delivery, and exceeding estimated costs [1]. Some of the problems encountered during teaching the SE course are related to its title while others are related to its contents. We begin with those related to its title.

2.1 Course Title

One of the first problems faced during teaching the course was to explain its title and why the word "engineering" was in its title. The IEEE Computer Society's Software Engineering Body of Knowledge defines Software Engineering as the: " application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, and the study of these approaches; that is the application of engineering to software" [2]. This means that "engineering" is the application of a "systematic, disciplined, quantifiable approach". However, according to The American Engineer's Council for Professional Development, "engineering" is: "the creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same with full cognizance of their design; or to forecast their behavior under specific operating conditions; all as respects an intended function, economics of operation and safety to life and property" [3].

The relationship between the two definitions is not, and cannot be, tight. Software development is very different

from engineering, for example: software is intangible meanwhile engineering applications are tangible, the existence of many programming languages with many features makes it possible to have many solutions to the same problem, and the reusability and reproduction of a solution to a problem in many other problems makes it hard to assess the effort involved. Other resources [4] point out that software development should follow an engineering paradigm. This means that there is a standalone "engineering paradigm" which has well defined steps. However, top ranked results returned from searching the Internet with various search engines for that term returned resources having "software engineering paradigm". That was really confusing to students. The software development process must follow the engineering paradigm which itself does not have a clear definition. A good solution for this problem is to accept Alistair's claim that [5]: "The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering". Specially that the term "software engineering" first appeared in the 1968 NATO Conference on Software Engineering, and which was aimed to stimulate software professionals and researchers to respond to software crisis at that time [1].

A second justification for using the word "engineering" in the title is to relate it to "Systems Engineering". According to The International Council on Systems Engineering (INCOSE), Systems Engineering is: "An interdisciplinary approach and means to enable the realization of successful systems" [6]. An expanded definition for systems engineering is given by the National Aeronautics and Space Administration (NASA) [7]: "System Engineering is a robust approach to the design, creation, and operation of systems. In simple terms, the approach consists of identification and quantification of system goals, creation of alternative system design concepts, performance of design trades, selection and implementation of the best design, verification that the design is properly built and integrated, and post-implementation assessment of how well the system meets (or met) the goals" . So depending on the previous definitions one may justify that the term "engineering" in software engineering was either borrowed from system engineering to mean "an interdisciplinary approach and means to enable the realization of successful software systems", or to denote that if an engineering system has a software component, and most probably it would, then Software Engineering "is a robust approach to the design, creation, and operation of software systems. In simple terms, the approach consists of identification and quantification of software system goals, creation of alternative software system design concepts, performance of design trades, selection and implementation of the best design, verification that the design is properly built and integrated, and post-implementation assessment of how well the system meets (or met) the goals". In all cases, in our opinion, this does not make Software Engineering an engineering discipline.

The some may agree or disagree with the above trials to explain the title of the course. However, we believe that there has been much room for trials because the 1968 NATO Software Engineering Conference did not give an explanation. according to Alistair [3]: "despite having the term as a focal point for the conference, the participants showed little understanding of either the term "Software Engineering" or engineering in general, and provide little guidance as to just what readers are supposed to infer from the term "Software Engineering." Alan Perlis' keynote speech contains the following: this is the first conference ever held on Software Engineering and it behooves us to take this conference quite seriously since it will likely set the tone of future work in this field in much the same way that Algol did. We should take quite seriously both the scientific and engineering components of software, but our concentration must be on the latter. Unfortunately, that is all he offers on the intention of the term."
Questioning whether software engineering is an engineering discipline at all is not new [7, 8, and 9].
Also, the teaching of Software Engineering as a subject is in continuous debate [10, 11]. It is not the goal of this paper to add to the doubts about the Software Engineering as a discipline or its education, but rather to find solutions to problems encountered during teaching the course. We find that Alistair's justification that the term was deliberately chosen as being provocative is an acceptable solution to the confusing title of the course.

2.1 Course Contents

Other problems encountered in the course were related to the course contents. Browsing syllabi of many software engineering courses, including ours, would lead to the conclusion that most of them have the following contents in common:

a- Introduction to software engineering
b- The software development process and software life cycle
c- Requirements specifications
d- Analysis and design (structured, object oriented approaches and UML)
e- Implementations, testing, maintenance and reliability
f- CASE tools
g- Other topics (e.g. project managements)

Problems related to (a) above would mainly involve the title and this has been dealt with earlier.

The software development process and software life cycle usually introduces students to the waterfall model, iterative models (e.g. spiral model), agile model, extreme model and rational united process. The waterfall model is well defined and the differences between this model and other models are clear. However, the differences between the other models are not that clear and could be confusing. For example, it is difficult to explain and highlight rigid differences between the spiral and agile models. Both are incremental and iterative. Both work in order of risk. The difference may be in the scope. While the spiral focuses on big design from the beginning and is recommended for large projects, the agile focuses on one increment at a time and may work for small projects. That difference is not really sharp to require two names for almost the same model. It was going to be easier if agile was considered a special case of the spiral model. Also, it is not clear what is meant with big and small projects, this is proportional. If the course delves into the discussion of the Extreme Programming (XP) model / technique then more confusion is added to the course as follows: The PC magazine says about XP that "it is based on a formal set of rules about how one develops functionality such as defining a test before writing the code and never designing more than is needed to support the code that is written" and "XP is designed to steer the project correctly rather than concentrating on meeting target dates, which are often unrealistic in this business" [12]. But is not that what software developers need? Just to design what is needed for coding and to steer the project correctly? If so, then why the need for other models? Even more, TechTarget [13] claims that: "Kent Beck, author of Extreme Programming Explained: Embrace Change, developed the XP concept. According to Beck, code comes first in XP". But this contradicts what we have been teaching students that software engineering is concerned with the careful analysis and design so that the coding phase goes smoothly. Now, we teach them that code comes first. Furthermore, according to Don Wells [14], XP "has already been proven to be very successful at many companies of all different sizes and industries worldwide". Again, if XP is the perfect model for all different sizes and industries then why trying other models? On the other hand, the some suggest that XP is waning [15]. While most literature suggests that XP is a special case of agile, Extreme Programming (XP) happens to be the most well-known of agile methodologies [16]; others suggest that agile itself is only an implementation of the spiral model [17]. The point here is that there is no consensus on the relationship between the different models and there is no clear recommendation on when to use each. We transfer this confusion to students in our teaching. Now, how about adding the Rational Unified Process (RUP) to the picture? We suggest not to overwhelm students with many techniques and models, but rather to introduce them to the waterfall model and the spiral model and we list the agile, pair programming, and Extreme programming as different implementation of the spiral model and focus on the XP since it seems to be working and we believe that students actually have been following this technique in their programming courses without actually realizing that it has the name XP. Once students learn a programming language they become enthusiastic to using it and start coding quickly. So, they actually design little and later and code first. Of course, we have to shape their skills in using these techniques, but it is the closer to them.

A third source for problems encountered was the topic of "Analysis and design (structured, object oriented (OO) approaches and UML)". This is due to the similarity between some of the tools used in the structured and object-oriented approaches. A student once asked why I should use use-cases, sequence diagrams and class diagrams when I can use the entity-relationship diagram I learned in the database course and the data flow diagram and process flow diagram which I have learned in other courses. The structured approach mainly uses the entity-relationship diagram (E-R) and the data flow diagram (DFD), whereas the object-oriented approach may use the UML including:

- Use case diagrams
- Class diagrams
- Sequence diagram
- Object diagram
- Package diagram
- Deployment diagrams
- State machine diagram
- Activity diagram
- Communication diagram
- Component diagrams
- Interaction overview diagrams
- Timing diagrams

Although there are differences between the structured (functional decomposition) approach and the OO approach, but there are also big similarities between some of their tools (e.g. the E-R diagram and the class diagram). This makes students ask why the E-R diagram is not part of the UML. An entity in the E-R diagram corresponds to the class in the class diagram. Attributes in the E-R diagram corresponds to attributes in the class diagram. Relationships between entities correspond to relationship between classes. Of course the latter have methods and

operations as well. But students wonder if the structured approach is supposed to be considered a completely different approach from the object-oriented approach whereas major tools are almost the same (or very similar) in both. The instructor focuses on the fact that a class diagram represents the behavior features of a system through the operations. A similar argument can be said about the similarity between the DFD and the sequence diagram or the activity diagram. Since the UML with its various diagrams are more comprehensive then we believe that it should be used directly without actually considering the E-R diagram and the DFD. A brief introduction to the structured approach may be considered but without delving into the tools.

4. Conclusions

We have identified and presented some problems encountered during teaching the course of software engineering with some brief and quick suggestions. We believe that most of these problems encountered are due following a traditional course syllabus that addresses both the structured and OO approaches in detail, and also for considering many diagrams that are part of UML. We believe that the course must involve extensive programming and many case studies to be interesting to students and to clarify to students that this course does not provide one proper solution to developing software, but rather various approaches could be adopted and that there is room for creativity. We are currently working on developing a new syllabus which addresses the contents problems raised in this paper in more details and which we expect to make the course interesting and more applied.

References

[1] P. Naur and B. Randell, Eds. Software Engineering. Report on a Conference held in Garmisch, Oct. 1968, sponsored by NATO

[2] SWEBOK executive editors, Alain Abran, James W. Moore; editors, Pierre Bourque, Robert Dupuis. (2004). Pierre Bourque and Robert Dupuis. Ed. Guide to the Software Engineering Body of Knowledge - 2004 Version. IEEE Computer Society. pp. 1–1. ISBN 0-7695-2330-7

[3] Science, Volume 94, Issue 2446, pp. 456: Engineers' Council for Professional Development

[4] A Brief History of Software Engineering. Online resource: http://www.comphist.org/computing_history/new_page_13.htm Retrieved October 17, 2010.

[5] Alistair. The end of software engineering and the start of economic-cooperative gaming. Online Resource: http://alistair.cockburn.us/The+end+of+software+engineering+and+the+start+of+economic-cooperative+gaming Retrieved Oct 1, 2010.

[6] Systems Engineering Handbook, version 2a. INCOSE. 2004.

[7] NASA Systems Engineering Handbook. NASA. 1995. SP-610S.

[8] Mahoney, Michael. 2004. Finding a History for Software Engineering, Online resource http://www.princeton.edu/~hos/Mahoney/articles/finding/finding.html Retrieved September 15, 2010.

[9] M. Shaw, "Prospects for an Engineering Discipline of Software", IEEE Software vol. 7, no. 6, Nov. 1990, p. 15.

[10] Parnas, D. L., Software Engineering Programs are not Computer Science Programs, IEEE Software, November/December, 1999, Vol. 16, No. 6, pp. 19-30.

[11] Demarco, T. Point-Counter Point: It Ain't Broke, So Don't Fix It, IEEE Software, November/December, 1999, Vol. 16, No. 6, pp. 67-69.

[12] PCMAG. Extreme Programming. Online resource: http://www.pcmag.com/encyclopedia_term/0,2542,t=XP&i=55075,00.asp Retrieved Sep18, 2010.

[13] Techtarget. Extreme Programming. Online resource http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci214366,00.html Retrieved sep15, 2010.

[14] Extreme Programming. : Extreme Programming: A gentle introduction. Online resource http://www.extremeprogramming.org/ Retrieved Sep13, 2010.

[15] Smith, Steve. Is Extreme Programming Dying? Is Agile Growing in Popularity? Online resource http://stevesmithblog.com/blog/is-extreme-programming-dying-is-agile-growing-in-popularity/ Retrieved Oct 1, 2010.

[16] Hutagalung, Wilfrid. 2006. Extreme Programming. Online resource http://www.umsl.edu/~sauterv/analysis/f06Papers/Hutagalung/#xp Retrieved Oct 18, 2010.

[17] Stackoverflow. Online resource http://stackoverflow.com/questions/253789/agile-vs-spiral-model-for-sdlc Retrieved Oct 1, 2010.

Osama Shata is an Associate Professor in the department of Computer Science and Engineering at Qatar University, Qatar. He has an extensive industrial, academic and administrative experience at both the postgraduate and undergraduate levels. His research interests began with intelligent database systems and knowledge base systems. As information technology became an integral component of any successful education process, his research interests focused on e-learning / distance education, electronic course delivery, curriculum development and integration, multimedia, HCI, curriculum design and development, and Accreditation.