Learning and Instructional Issues in Software Engineering

Mike Murphy
mmurphy@spsu.edu
+01.770.528.4982

Rich Halstead-Nussloch
rhalstea@spsu.edu
+01.770.528.5509

Department of Computer Science
Southern Polytechnic State University
Marietta, GA 30060-2896 USA

ABSTRACT

Issues associated with the teaching and learning of Software Engineering and, more specifically, with forming Software Engineering teams are explored. A simple survey was constructed to assess the main factors for further research into teaching and learning styles. As an initial step, the survey was tried with graduate students to explore their perceptions and preferences. The results of that survey were very informative about learning in the classroom. They also provide a view toward future steps in understanding and creating a learning-centered environment for our students and in generating ideas for better practices in forming Software Engineering teams.

Keywords Learning styles; instructional issues; surveys.

1 INTRODUCTION

This position paper is the result of a series of conversations and investigations by the authors regarding the role of learning and teaching in the field of Software Engineering. This discussion has benefited additionally from our early-career experiences on software development teams at, e.g., NASA and IBM. Even now, the situation is like a series of sight-impaired people exploring the various local features of an elephant. It is very difficult to get a feel for the whole, and that is probably as it should be.

Within the educational domain, we see at least two reasons for this difficulty. First, all educators are faced with an enormous number and variety of very good conceptual models and theories for learning and teaching in the classroom. These concepts and models, although informative, are numerous and complex, leading to difficulty in making choices among them for an educator's focussed attention. Second, educators are faced with a large distance between these concepts and models and their application in the classroom. By their nature these models and concepts are abstract, and not subject to direct application. Of course the direct application of the concepts in formulating Software Engineering teams is equally problematic.

Despite the difficulty and its reasons, let's start at the beginning of our quest as best we can recall it. One of the authors presented a paper [1] at the Twelfth Conference on Software Engineering and Training (CSEE&T'99) relating experiences with an innovative approach to teaching software project management to graduate-level Software Engineering students. The presentation was well received and generated much discussion. The other author has developed a text book [2] for a Computer Science and Software Engineering graduate course in research methods. In addition to fundamental research to extend knowledge, this course covers practical research techniques and study designs for answering important empirical questions in Software Engineering. Development team building and configuration is a prime example of such an empirical question. When we recognized many common issues in both of our courses, we decided to empirically explore aspects of effective learning and teaching of Software Engineering in a broader context.

2 TEACHING, LEARNING, & LEARNING STYLE

A number of issues arise when we explore how students learn. Along with many other educators, we have found

a better understanding of teaching, learning, and the classroom experience come through looking into learning styles. Many software development firms are finding value in utilizing similar means to build development teams. Some of the conceptual approaches and models that come to mind are Temperament Types (Jung-Myers-Keirsey), Taxonomy (Bloom), Categories of Engineering Design Knowledge (Vincenti), Knowledge-Generating Activities (Vincenti), relate-create-donate philosophy (Shneiderman), and even the pioneering work of Dewey, Piaget, and Papert.

Anyone who is engaged in the scholarship of teaching [3] will find value in looking at these sources. They can make teaching an even richer and rewarding activity. Many software development firms are utilizing similar means to formulate better development teams. We have found that issues arise in bridging the gap between these models and classroom action and in configuring software development teams. We discuss these below.

The Temperament Type factor relates to the classification of temperament along 4 lines:

- extraversion (E) vs. introversion (I)
- sensing (S) vs. intuition (I)
- thinking (T) vs. feeling (F)
- judging (J) vs. perceiving (P)

From these come generalized types:

- guardians (SJ)
- artisans (SP)
- idealists (NF)
- rationals (NT)

Keirsey [4] provides a web site with a simple classification quiz so the data for this is relatively accessible. There are even some implications for learning styles depending on one's type. The more interesting issue, which has been explored by some, is what mix is best for collaborative work as in a group or on a team.

Bloom's Taxonomy [5] has long been used to classify levels of abstraction for learning units and examination questions:

- knowledge (partly precondition, partly defining and listing)
- comprehension (partly precondition, partly experiential)
- application (learning by doing)
- analysis (thinking)
- synthesis (problem solving)
- evaluation (reflection)

SWEBOK, the SoftWare Engineering Body Of Knowledge project [6], includes this classification related to units of study within major chunks of Software Engineering. The difficulty is in how to address learning effectiveness to achieve a desired level of abstraction.

Vincenti's Categories of Engineering Design Knowledge [7] are also used in the SWEBOK project for critical insight into the overarching goal of Software Engineering—good design, since good design reflects requirements and leads to effective and efficient implementation of a solution to a problem, the Holy Grail of problem-solving. The categories are:

- fundamental design concepts
- criteria and specifications
- theoretical tools
- quantitative data
- practical considerations

- design instrumentalities

The issue here is generally categorization after the fact.

Vincenti's Knowledge-Generating Activities [8] may be of somewhat more significance with respect to learning:

- transfer from science
- invention
- theoretical engineering research
- experimental engineering research
- design practice
- production
- direct trial

The issue then is how these translate into learning activities for the student of Software Engineering.

Shneiderman's relate-create-donate philosophy [9] builds on the ancient Chinese proverb: "I hear and I forget, I see and I remember, I do and I understand." Carefully selecting pieces of this, we can note:

- I hear and I forget—problems with lecture style of presentation
- I see and I remember—the role of visualization in learning
- I do and I understand—the role of participation in learning
- Relate—the key to teams and the whole being greater than the sum of its parts

These concepts and models may well provide a clear, if somewhat simplistic, looking glass into aspects of successful learning.

3 APPLICATION IN THE CLASSROOM AND ON THE JOB

When one pursues the scholarship of teaching, the conceptual models of learning and learning styles are necessary, but not sufficient. These concepts must be applied in the classroom. In Software Engineering, learning is action oriented--students must actively design and engineer software to be successful. This is distinct from education in other disciplines, such as the humanities, math, and even the sciences, where designs and related products are not normally produced. A major question for Software Engineering education is how to apply the rich conceptual information about learning and learning styles in the classroom and to form development teams at work. In our case we wondered how to boil down the large volume of good conceptual material into a broth that was rich for our Software Engineering classes and our students' application on the job.

4 OUR FIRST STEP--CONDUCT A SURVEY

Despite being somewhat overwhelmed by the volume and abstract nature of these issues about learning, it became apparent that we may be looking at an NP-complete problem. Being stalwart Computer Science and Software Engineering educators, we started with a fairly simple experiment that points the way to future exploration of learning and Software Engineering. As a first step we decided to try a quick and efficient survey in our classrooms. Surveys are often used to assess learning style in other discipline's classrooms. We wanted to see if the use of such a survey in our classrooms would bring benefits.

At approximately midterm of the Fall Semester 1999, a survey was administered to students in 3 graduate classes taught by the authors: Software Project Management (PM), Research Methods (RM), and Human Factors (HF). Utilizing the learning concepts, models, and theories discussed above, we developed a questionnaire that covered what we assessed as the major factors for Software Engineering education. Our

objective was a diagnostic instrument that could be used to assess quickly and painlessly the "state of learning" in the Software Engineering classroom. The questions included:

- demographics (gender, age, nation of birth, native language, marital status, student status, major)
- work-related material (job status, IT connection, longevity)
- current class situation using a sliding scale between opposites (good project management vs. luck, independent work vs. group work, intuitive vs. objective, controlled vs. chaotic, and art vs. science)
- attitudes toward learning styles using an agreement scale (collaborative, asynchronous, lecture, visually rich, rich in sound, tactile/doing, stress)
- attitude toward the survey using an agreement scale

All of the students were quite cooperative in filling out the surveys and had a generally positive attitude toward the survey itself. The students were assured that their responses were anonymous and that there were no preferred responses. The survey questionnaire required between 5 and 10 minutes for completing. In all we surveyed 52 respondents.

Our interest was mostly in perception of the current class situation and the attitudes toward learning. The results in general lend insight through observable trends rather than the usual statistical analyses. This tells us that experimental design for the future needs careful attention to give a deeper degree of insight. The following table summarizes the profile that emerged.

Table 1. Observed descriptors for the three courses.

PM	**RM**	**HF**
Good project management	Good project management	Good project management
Control		
Science		
Collaborative	Collaborative	Collaborative
Asynchronous	Asynchronous	Asynchronous
Lecture	Lecture	Lecture
Visual	Visual	Visual
Tactile/Doing	Tactile/Doing	Tactile/Doing
No Stress	**No Stress**	**Stress**

The first 3 rows indicate the students' view of their current class situation. All three classes clearly described the class situation involved good project management more than luck. The PM class more definitively specified the class situation as involving control (about 5 of 6) more than chaos. The RM (about 3 of 4) and HF (about 2 of 3) classes, although not different statistically from the PM class, showed a smaller frequency of choosing the control descriptor for their class situation. A similar profile was seen on the science versus art measure where about two of every three in the PM class characterized the situation as more science than art, where about one of two do so in the RM and HF classes. This profile is also not statistically significant.

That RM and HF didn't clearly express a view on control vs. chaos and art vs. science is interesting and consistent with differences among the courses. In the PM course, a major emphasis is placed on maintaining proper management control of the situation, while the RM and HF courses emphasize acquiring appropriate knowledge and skills. The PM course emphasizes science over art through the constant requirement for measurement. The RM and HF courses, while both based in science, emphasize the artistic facets of rhetoric and presentation (RM) and user-interface design (HF). So, these results are both interesting and understandable.

The last 6 rows of Table 1 were attitudes toward learning ("I learn well when…"). The most interesting item is the preference for the students in HF for stress where the other 2 classes preferred no stress. A chi-square analysis, done on a collapsed table to increase cell n shows this difference to be statistically significant (chi-square is 7.323, df = 2, $p<0.05$), indicating the HF class has a higher tolerance or even a higher *need* for stress.

In sum, the survey has shown itself to be easily completed and at the same time to produce reasonable information for instructor action. For example, it would make sense to ensure the HF course had the pressure of deadlines, while downplaying those pressures in the other two courses.

5 NEXT STEPS AND SPECIFIC GOALS FOR OUR WORKSHOP PARTICIPATION

Our plan for the future is to

- Improve our survey and its implementation design
- Expand the number of classes surveyed
- Seek greater variation in content and nature of the classes
- Focus on learning styles vs. teaching styles
- Investigate implications for Software Engineering team formation in an industrial setting

There are 3 major factors that needed to be explored both individually and as they interact:

- Learning styles and preferences (student perspective)
- Course content
- Teaching styles

These may seem obvious, but the real issues are the complex interplay of the 3 factors—somewhat of a three-dimensional chicken and egg situation. We see these as empirical questions and therefore our teaching and also the practice of our students in their teamwork can greatly benefit from expert presentations and lively discussions of the workshop. Since we want to take this empirical research to the next level, we will aim our participation towards specific goals that include:

- Empirical mapping and comparison of the Software Engineering educational and application environments
- Differences/similarities in the role of collaboration in the learning and working environments
- The view of other disciplines toward learning style versus instructional style
- A critique of our instrument to indicate directions for improvement, establishing norms, etc
- Identifying additional empirical methods to be included in the textbook [2]
- Establishing a rapport with the international Software Engineering education community for purposes of sharing knowledge and building a bridge to future collaboration.

REFERENCES

1. Michael G. Murphy, Teaching Software Project Management: A Response-Interaction Approach, *Proceedings of the 12th Conference on Software Engineering and Training*, 1999, pp. 26-31.
2. Richard Halstead-Nussloch and Bob Harbort, *Research Methods in Computing*. Simon and Schuster, 1999.
3. Boyer, E.L., *Scholarship Reconsidered*. San Francisco: Jossey-Bass, 1990.
4. http://www.keirsey.com/
5. http://www.coun.uvic.ca/learn/program/hndouts/bloom.htm
6. http://www.swebok.org/
7. Walter G. Vincenti, *What Engineers Know and How They Know It*, John Hopkins, 1990, pp.207-225
8. Walter G. Vincenti, *What Engineers Know and How They Know It*, John Hopkins, 1990, pp.225-237
9. Ben Shneiderman, Relate-Create-Donate: A teaching/learning philosophy for the cyber-generation, SIGCSE'98

Mike Murphy, Ph.D., is currently Interim Vice President for Information Technology (CIO) and Professor of Computer Science at Southern Polytechnic State University in Marietta, GA, USA. He regularly teaches Software Project Management in the Master of Science in Software Engineering program at SPSU. Current interests include Software Engineering curriculum and related issues of learning and teaching styles.

Rich Halstead-Nussloch, Ph.D., is currently Director of the Office of Sponsored Projects and Associate Professor of Computer Science and Industrial Engineering Technology at Southern Polytechnic. He regularly teaches Human Factors and Research Methods in the Master of Science in Software Engineering, Master of Science in Computer Science, and Master of Science in Quality Assurance programs at SPSU. Current interests include Software Engineering curriculum, related issues of learning and teaching styles, and methods of empirical research in computing and Software Engineering. He is engaged in research into software design.