AC 2007-145: A NEW PARADIGM TO IMPROVE COMPUTER EDUCATION FOR ENGINEERING STUDENTS: APPLYING INDUSTRY-BASED SOFTWARE DEVELOPMENT CYCLE INTO PROGRAMMING PRACTICES

Wangping Sun, Oregon Institute of Technology

Dr. Wangping Sun is an assistant professor at Oregon Institute of Technology. He holds a bachelor's degree in mechanical engineering, a master's degree in software engineering, and a PhD degree in industrial engineering. He has ten years of work experience in industry and four years of work experience in information technology. He is a member of SME, IIE, and ASEE.

A New Paradigm to Improve Computer Education for Engineering Students: Applying Industry-based Software Development Cycle into Programming Practices

Abstract

Computer Programming for Engineers (or Introductory Computer Education for Engineering Students) is a fundamental engineering course in many universities. However, in the public domain, there has not been enough research on how to improve programming practices (programming labs and homework assignments) of the students. What paradigm should be followed when the students are programming? How is this paradigm helpful for students' learning? These questions are yet to be answered. In this paper, the author will introduce a new approach to administer programming practices of the students. The benefits and implementation steps of this new paradigm will also be discussed.

Keywords: Computer Education; Computer Programming; Engineering Education; Professional Development; Software Development Cycle.

1. Introduction

Computer programming is an essential and integral part of any engineering program [1]. Engineering students must be able to use a variety of rapidly changing computing systems and tools to solve an ever-expanding range of problems across disciplines [2]. Engineering schools offer the computer programming course in freshmen or semaphore year in engineering or engineering technology program [3,4].

In our institute, Computer Programming for Engineers (ENGR 266) is an introductory computer programming course for non-computer-science sophomore engineering students. In this course, the students meet three times weekly (two one-hour lectures, and one three-hour lab) to learn essential computer programming skills by using MatLab 7.0 and Visual Basic Application (VBA) in Excel. Engineering problems in manufacturing, mechanical engineering, electrical engineering, dynamics, statics, physics, and math are given to the students as programming practices (programming labs and homework assignments). The textbooks being used are:

- *MATLAB: An Introduction with Applications, 2nd Edition,* by Amos Gilat
- *Excel Programming: Your visual blueprint for creating interactive spreadsheets, 2nd Edition*, by Jinjer Simon.

When the author of this paper taught the course for the first time, all the programming practices were individual-based. The students were given some computational questions and required to finish the coding individually within certain amount of time. However, this "conventional" approach, which is still widely used by a lot of other engineering schools, created many

problems. Each time in the lab, almost 70% of the class struggled to make a Matlab or VBA program work in the required time. The instructor was busy helping the students to debug their programs (this class does not have a teaching assistant). During the office hours, the students frequently visited the instructor to request for hints or even solutions to fix a bug in their homework and they often complained that the homework was too difficult. The evaluation on this class showed that the learning effectiveness was not satisfactory because students said that they were still lack of confidence in programming skills even after taking the course.

To address these issues, research was conducted for pedagogical improvement of the course. A new approach, industry-based software development cycle (IBSDC), was developed and piloted starting from fall 2006. The initial result looks encouraging. In the following sections, the paper will first review the recent literature in teaching Computer Programming for Engineers. Then, it will introduce IBSDC and explain the benefits of bringing this paradigm into computer education for engineering students. Next, the implementation of IBSDC will be described. Following that, the evaluation on IBSDC will be discussed. Finally, in the last section, the conclusions will be drawn.

2. Literature review

Quite a few educators have done research on how computer education can be offered to engineering students. The research has been focusing on three areas: 1) why to teach: the objectives of the course; 2) what to teach: languages, tools, and topics to be covered in the course; 3) how to teach: approaches to effectively offer the course.

2.1 Why to teach

The objectives to offer the introductory computing class to engineering students are basically similar from school to school, which can be summarized as follows [3,5,6-9]:

- Providing students with skills necessary to begin a career in engineering discipline;
- Ensuring that students have sufficient programming background for solving problems in engineering;
- Introducing engineering applications in different disciplines by using structured programming;
- Using tools for engineering analysis, calculation, and graphical display;
- Understanding programming fundamentals, including the essence of object-oriented programming;
- Opening the door for further study and specialization in computer science.

The abilities that the students are expected to obtain or strengthen from the course include: 1) interpretation of the problem, 2) design of solution strategy, 3) problem solving by numerical computation, and 4) development of appropriate documentation.

2.2 What to teach

The majority of the schools are switching from traditional languages, such as Pascal, Fortran, C, Visual Basic, C++ or Java, to Excel/VBA and Matlab. There are different reasons respectively for using Excel/VBA and Matlab.

For Excel/VBA, four major factors decided its popularity [1,5, 6]. Firstly, engineering schools are trying to avoid the "soon-forgotten" syndrome as observed in traditional programming classes (students learn a programming language, but do not use it routinely to solve problems in their engineering courses, and so, their programming skills get rusty quickly). Secondly, Excel spreadsheets are widely used in industry. Thirdly, through Exel/VBA, students are able to differentiate themselves from other people who are limited to built-in features with Excel. Fourthly, once the students learn one programming language, migrating to other languages becomes easy. For many engineering disciplines, it is possible to prepare students for industrial practice while meeting academic expectations for an understanding of programming concepts by using a spreadsheet in combination with Visual Basic [10].

For Matlab, two major reasons decided its popularity [10,11]. Firstly, the students can use it to solve engineering problems and get some programming skills without previous programming experience. Secondly, the data handling is more visible and easier to comprehend for students.

The topics covered in Excel/VBA are listed as follows [1,3,6-8]:

- Variables, logical operators;
- Arrays;
- Sub-procedures and functions, parameter passing/returning;
- Iterations (loops);
- Decisions (if-then-else, select-case statements);
- Built-in and user defined functions;
- Graphic user interface development;
- Spreadsheet, statistical tools, matrix operations, non-linear system of equations;
- Concepts of object-oriented programming.

In teaching Matlab, the following topics are covered [5,6,11]:

- Matlab editor, Help menu and tutorials;
- Vectors , array, and matrices;
- 2D, and 3 D graphics plotting;
- Functions, parameter passing and value returning;
- Conditional statements and loops;
- Symbolic toolbox;
- Linear algebra;
- Polynomials and interpolation techniques
- Input/output, *.m files, inline functions;
- Numerical integration, difference and differential equations;

- Image, sound and signal processing.

Instead of Matlab, some other schools introduce other software tools [4,5,9,12]. These tools include: Mathcad, EES (Engineering Equation Solver), Mathematica, and Maple.

2.3 How to teach

Computer science department in some universities is still responsible for teaching programming skills to engineering students [3,9].

For other schools that offer the course in engineering disciplines rather than computer science, almost all of them use problem-driven approach [10]. With this approach, an engineering or scientific problem is presented to the students by the instructor. Then, mathematical description of this problem is provided in the form of a set of equations or data tables. After that, the computational method is introduced and applied to the problem. At last, students are asked to consider variations on the applications that use the techniques of interest.

As noted by Covington and Benegas [8], the current methodologies for teaching programming skills depend heavily on syntax-driven approach. They argued that this approach places undue early emphasis on language syntax and not enough on understanding and problem solving. They proposed schema-driven approach as an alternative. The essence of their approach is to help the students to recognize the solution pattern (or "schema") to a class of problems. With schema-driven approach, the students spend relatively more time thinking about problem characteristics and problem solving solution patterns. The students are then directed to focus on the details of the language syntax. Four steps are taken to implement schema-driven approach:

- Presentation of several problems from a similar class of problems;
- Explanation of general similarity of solution approach;
- Introduction of relevant programming language features for this class of problems;
- Integration of the solution approaches with the programming language features.

Computer programming is one of the courses that students typically struggle with motivation and enthusiasm. Traditional programming is dry, boring and irrelevant [4]. Many techniques have been attempted to stimulate interest [13]. To make lectures friendly and inviting, pop music or futuristic space music is played in some schools before the lecture and during the break [14]. In an attempt to provide extra incentive, some schools required their students visualize the problem solutions by animation [4,15]. Some schools asked the student to make presentations by using Power-Point, and the presentations were videotaped and critiqued [5,6]. List-serv is also used to help the students to post questions and arouse discussions [16].

2.4 A significant issue to be addressed

Computer programming education is not as mature as the teaching of the sciences and engineering topics [8]. All the recent research has not given enough emphasis on an important part of the course: student programming practices (programming labs and homework assignments). In the current literature, it seems that the programming practices are lack of good organization,

sufficient guidance and strong commitment to teamwork. Rowley and Bazzoli mentioned that student participation in small groups was periodically called for to solve examples from the lecture, and some homework required the students work interactively [14]. But, no details were available on how the group events were conducted. Coronell mentioned that in their labs, the students were free to work together and free to ask questions [7]. Whereas, there was no further information on these collaborations are managed. Collura et al. briefly said that in their labs the more advanced students need to help their neighbors who are having difficulty [10]. However, the effectiveness of this learning style for the students seeking for help is questionable. Navaee and Naraghi gave the project questions to the students and then let them manage their projects on their own [1,16]. More control by the instructors on the programming projects is obviously needed.

According to the author's observation, programming practices often account for at least 70% of the efforts the student puts into the course. Only through programming practices can the students clarify the ambiguities, strengthen problem-solving abilities, and improve their programming skills. However, the issue of lack of good administration in programming practices negatively impacts learning effectiveness:

Firstly, due to lack of good organization, a programming practice is likely to become an individual activity. A frustrating learning environment can be created unexpectedly, in which some students will be negatively influenced. As observed by the author, at the end of each individual-based programming lab, there was always 10% - 15% of the class who got really frustrated. They either asked for extra time to finish the assignment or quit further working on the unfinished work.

Secondly, due to lack of enough guidance, it is hard for the students to capture the major problem solving skills. This can be reflected by the insufficient confidence of the students in their programming capabilities. As experienced by the author, even after taking the programming class, the students would often provide feedback like:

- "[I] need more labs."
- "[I] need more real-life applications."

Thirdly, due to lack of strong commitment to teamwork, workload allocation for group projects is very unbalanced. As observed by the author, some "capable" students are likely to predominate the team and learn more than the "less-capable" students on the same team. The performance gap between "capable" and "less-capable" students gets wider when more group activities are held.

3. Industry-based software development cycle

The author of the paper proposes a solution, industry-based software development cycle (IBSDC), to address the issue described in Section 2.4.

3.1 Introduction of IBSDC

IBSDC is a process for professional software development. The software development cycle may vary from one field (say, medical industry) to another (say, telecommunication industry). But, four phases, as shown in Figure 1, exist in almost all cases (for more detailed information about software development cycles, please refer to reference 17).

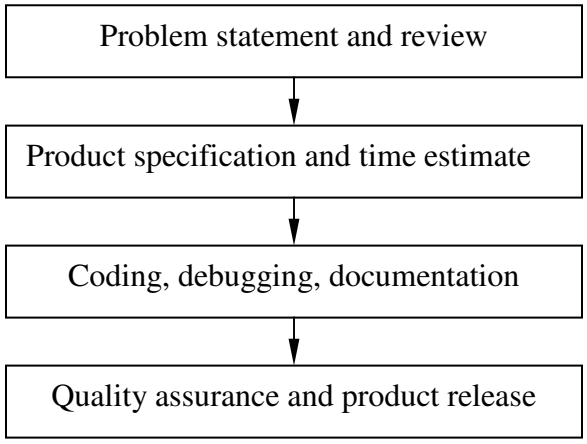


Figure 1. Four phases in IBSDC.

In the first phase, when a new project comes in, the developer analyzes the problem statement and obtains an accurate understanding of the project objectives. Communication between the developer and the end-user is necessary to clarify any ambiguities. In the second phase, the developer develops product specification in the format of flow chart, pseudo code, algorithm or formal language. The specification depicts the design of the project. With this scheme, the developer estimates how much time is needed for this project. Time estimate is important to guarantee on-time delivery of the product. Next, in the third phase, the developer converts the design into lines of code. In the coding process, the developer needs to write complete documents to show how the problem is solved. Comments in the code should be as detailed as possible. Good documentation is important for software testing and future enhancement. Lastly, in the fourth step, the code will be tested against the boundary conditions and other test cases to see if the software works reliably in all possible circumstances.

In IBSDC, the developers' goal is straightforward: solve project problems and finish the coding in the required time. The developers working on the same team have the same goal. Pointless "trial and error" practice is regarded as a waste to be avoided. The development is accomplished in a cooperative environment. If one developer gets struck, the other team members, no matter if they work on the same project, must help.

3.2 Benefits of applying IBSDC

IBSDC is an attractive option for improving student programming practices. To use IBSDC provides an opportunity for the students to practice professionally and feel comfortable with it.

This is helpful for the students to step into industry smoothly in the future. Bringing IBSDC into student programming practices can improve the learning effectiveness in several ways:

Firstly, it creates a collaborative environment for the students to work with and learn from each other. Research shows that this environment improves learning efficiency, communication skills and teamwork [18,19].

Secondly, it is a problem-based and project-oriented approach, which is one of the most effective approaches in engineering education [20]. It encourages the students to think deeper and learn more.

Thirdly, the students will have equal opportunities to learn. In IBSDC, each student has different programming assignment and the student's performance is evaluated individually by the instructor. The workload unbalance can be reduced.

Fourthly, but not lastly, it is not a "quick-and-dirty" process. It requires the students program by following standard rules and conventions. It is a strict but helpful process to obtain good programming habits.

4. Implementation of IBSDC

As illustrated in Figure 2, IBSDC in programming labs is implemented in five steps. For the homework assignment, the implementation process is similar, but the time lengths may vary.

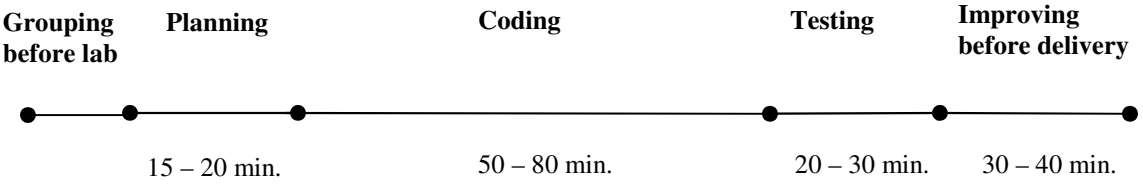


Figure 2. Implementation of IBSDC in programming labs.

The first step takes place before a lab begins. The students pair with each other to form groups (if the class enrollment is an odd number, one group will have three students). The students are encouraged to change partners each time so that they can have opportunities to work with different people. After the group is formed, two sets of problems with similar difficulty level are given to the students. Each student takes one set respectively.

The second step is to make problem-solving plans. After receiving the programming assignments, each group spends 15-20 minutes to study the problems and develop the pseudo code, flow chart or other plans to solve the problems. Then, the students will estimate how much time they need to finish the coding. Since the students may not have enough programming

experience yet, the instructor will provide a time guideline for them, based on which they may provide their own time estimate.

The third step is to write and self-test the code individually. Coding needs to be finished in the time line estimated by the second step. Depending on the problems, the coding process ranges from 50 – 80 minutes. While coding, the students are encouraged to follow the programming conventions provided by Johson [21] and Firesmith [22]. Whenever the students get stuck in programming, they need to ask their partners for help first. If the partners cannot answer the questions either, the instructor will help.

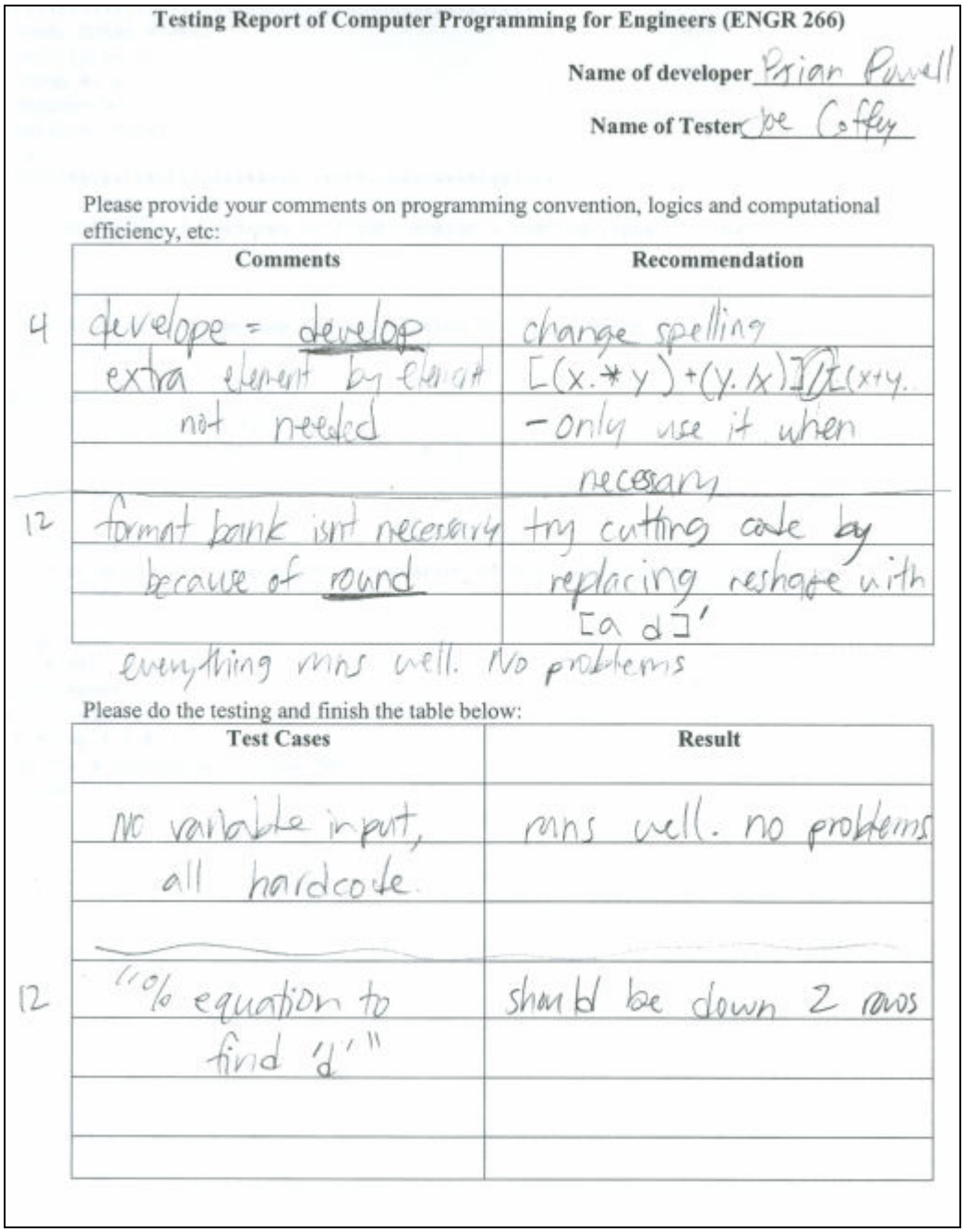
The fourth step is to test the code by group partner. In this process, the students in the same group exchange their code and make cross-testing. Each student checks to see if 1) the code has sufficient comments, 2) the programming conventions have been followed, 3) the programming logics are correct, and 4) the computational efficiency is good. After that, the tester develops test cases to see if the code can reliably work, especially at the boundary conditions. Finally, suggestions and testing results will be gathered in a testing report (as shown in Figure 3) and given back to the partner for code modification and improvement.

The last step is to improve the code before its submission. After receiving the testing report from the group partner, the student will follow the report to improve the program until the final version is available to be submitted to the instructor.

To ensure the quality of IBSDC, the programming assignment of each student is graded upon four factors:

- Code correctness (40%)
- Quality of testing report (30%) made for the group partner
- Obedience to convention with sufficient comments in the code (20%)
- On-time delivery of the code (10%)

The percentage in parentheses represents the weight of each criterion in grading. The allocation of the weights ensures that the students 1) treat both their own programming assignments and their partner's seriously, 2) follow the programming conventions, 3) write good documentation, and 4) submit the work on time.



Testing Report of Computer Programming for Engineers (ENGR 266)

Name of developer Brian Powell

Name of Tester Joe Coffey

Please provide your comments on programming convention, logics and computational efficiency, etc:

	Comments	Recommendation
4	develope = develop	change spelling
	extra element by each	[(x.*y)+(y./x)]/[(x+y.
	not needed	- only use it when necessary
12	format bank isn't necessary because of round	try cutting code by replacing reshape with [a d]'

everything runs well. No problems

Please do the testing and finish the table below:

Test Cases	Result
No variable input, all hardcode	runs well. no problems
12 "% equation to find 'd'"	should be down 2 rows

Figure 3. A testing report from cross-testing.

5. Evaluation on IBSDC

The benefits as described in Section 3.2 were all witnessed when IBSDC was implemented. In addition, one more difference between IBSDC and traditional programming approach was also noticed. IBSDC requires cross-testing between partners. According to the author's observation,

the students become really sharp when they review the other's code. This sharpness is a big incentive for the students to improve their programming skills and the quality of their work.

The effectiveness of applying IBSDC is evaluated by two criteria, based on the recent research achievements [23-25]:

- Performance of the students
- Confidence of the students

To evaluate the performance of the students, the grades were compared between the students who used IBSDC with those who did not in the previous terms. There is about 10% improvement in the average grade for the students who used IBSDC.

To evaluate the confidence in programming skills, the students were surveyed at the end of the academic term. Two questions were asked:

- Do you have enough confidence in using VBA/Excel and Matlab to do engineering computation in the future?
- Do you think that that IBSDC is an effective way for your learning?

Two thirds of the students reported that they are confident in their programming skills. Three fourths of the students agreed that IBSDC is effective.

6. Conclusions and future work

IBSDC is a new approach for engineering students to learn computer programming. It improves student programming skills and enhances cooperation and teamwork among students. More importantly, it makes the learning process an integral part of professional development for the students.

For today's engineers, the knowledge about computer technology will be a key factor for success. How to help engineering students to better prepare for their professional life is an utmost task for the educators. Following the current research, the author will collaborate with other faculty members to enhance computer programming in more engineering courses. This effort will help to eliminate the "too-dependent-on-calculator" syndrome, which is another common phenomenon in engineering education.

References:

1. Naraghi, M., 2005, "VBA/Excel: an alternative computer programming tool for engineering freshmen," Proceedings of IMECE 2005, pp. 315-322.

2. Urban-Luran, M. and Weinshank D.J., 2001, "Do non-computer science students need to program?" International Journal of Engineering Education, Vol. 90, No. 4, 535-541.

3. Azemi, A. and Pauley, L., 2004, "Using Matlab to teach the introductory computer-programming course for engineers," Proceedings of the 2004 ASEE Annual Conference & Exposition.

4. Myszka, D., 2006, "Motivating students in an introduction to computing course by requiring animated solutions," Proceedings of the 2006 ASEE Annual Conference and Exposition.

5. Clough, D.E., Chapra, S.C., and Huvard, G.S., 2001, "A change in approach to engineering computing for freshmen – similar directions at three dissimilar institutions," Proceedings of the 2001 ASEE Annual Conference and Exposition.

6. Clough, D.E., 2002, "ChE's teaching introductory computing to ChE students - a modern computing course with emphasis on problem solving and programming," Proceedings of the 2002 ASEE Annual Conference and Exposition.

7. Coronell, D.G., 2005, "Computer science or spreadsheet engineering? An Excel/VBA-based programming and problem solving course," Chemical Engineering Education Journal, Vol. 39, No. 2, pp. 142-145.

8. Covington, R. and Benegas, L., 2005, "A cognitive-based approach for teaching programming to computer science and engineering students," Proceedings of the 2005 ASEE Annual Conference & Exposition.

9. Josephson, W., Kwon, K.C. and Vahdat, N., 2004, "A self assessment of computer science education in a chemical engineering curriculum," Proceedings of the 2004 ASEE Annual Conference & Exposition.

10. Collura, M.A., Aliane, B. and Daniels, S. and Nocito-Gobel J., 2004, "Learning the methods of engineering analysis using case studies, Excel and VBA - course design," Proceedings of the 2004 ASEE Annual Conference and Exposition.

11. Rosca, R., 2006, "Learning Matlab – just-in-time or freshman year?" Proceedings of the 2006 ASEE Annual Conference & Exposition.

12. Hodge, B.K. and Steele, W.G., 2001, "Computational paradigms in undergraduate mechanical engineering education," Proceedings of the 2001 ASEE Annual Conference & Exposition.

13. Said, H., Khna, F. 2004, "Towards using problem-based learning in teaching computer programming," Proceedings of the 2004 ASEE Annual Conference and Exposition.

14. Rowley, B.A. and Bazzoli, T.L., 2005, "Freshman engineering & computer science program at Wright State University," Proceedings of the 2005 ASEE Annual Conference & Exposition.

15. Bualuan, R., 2006, "Teaching computer programming skills to first-year engineering students using fun animation in Matlab," Proceedings of ASEE Annual Conference & Exposition.

16. Navaee, S., 2006, "Computer utilization in enhancing engineering education," Proceedings of the 2006 ASEE Annual Conference and Exposition.

17. Pressman, R.S, 2004, Software engineering – a practitioner's approach, 6th edition, McGraw-Hill Companies, Inc., New York.

18. Pimmel, R., 2001, "Cooperative learning instructional activities in a capstone design course," Journal of Engineering Education, Vol. 90, No. 3, p. 413-421.

19. Sun, W.P., 2007, "Introducing cooperative learning into a traditional mechanical engineering course," submitted to Journal of Technology Interface.

20. Sun, W.P., and Anderson, J., 2006, "Teaching plant design/material handling by using project-based approach," Proceedings of the 2006 ASEE Annual Conference & Exposition.

21. Johnson, R., 2002, Matlab programming style guidelines, available from: www.datatool.com/downloads/matlab_style_guidelines.pdf

22. Firesmith, D., 2000, "Visual Basic coding standards," available from: http://www.donald-firesmith.com/Components/WorkProducts/ImplementationSet/SoftwareComponent/CodingStandards/VisualBasicCodingStandards.doc

23. Greca, A., Jovanovic, V. and Harris, J., 2003, "Enhancing learning success in the introductory programming course," Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference.

24. McDowell, C., Werner L., Bullock, H.E. and Fernald J., 2006, "Pair-programming improves student retention, confidence, and program quality," Communications of the ACM, Vol. 49, No. 8, pp. 90-95.

25. Pejuan, A., 2005, "An experience of collaborative learning with students beginning engineering," Proceedings of the 3rd International conference on Multimedia and Information & Communication Technologies in Education.