*British Journal of Educational Technology* *Vol 38 No 3 2007* 405–415
doi:10.1111/j.1467-8535.2007.00705.x

# *Teaching software engineering by means of computer-game development: Challenges and opportunities*

**Nergiz Ercil Cagiltay**

*Nergiz Ercil Cagiltay is an assistant professor at the Software Engineering Department of Atilim University. She holds a PhD in instructional technologies from Middle East Technical University, Turkey. Address for correspondence: Dr Nergiz Ercil Cagiltay, Bilgisayar Muhendisligi Bolumu Atilim Universitesi, H14, 06836 Incek—Ankara—Türkiye. Tel: +90 (312) 586 83 59; fax: +90 (312) 586 80 90; email: nergiz@atilim.edu.tr*

**Abstract**
Software-engineering education programs are intended to prepare students for a field that involves rapidly changing conditions and expectations. Thus, there is always a danger that the skills and the knowledge provided may soon become obsolete. This paper describes results and draws on experiences from the implementation of a computer game-development course whose design addresses problems in software-engineering education by improving students' abilities in four areas: (1) problem solving; (2) the application of previously learned knowledge; (3) the use of independent learning; and (4) learning by doing. In order to better understand this course's effect on students' performance in a software-development project, I investigated 125 students' performance in a 1-year senior-project course. Results of this study show that the students who had taken the computer game-development course became more successful in the senior-project course than the students who had not taken it.

## Introduction

The importance of information technology (IT) in our daily lives is growing. The rapid development of technology affects business and industry. The wide usage of technology creates very complex situations that professionals in IT need to be able to manage (Callahan, 2003). As the environment changes, industry demands regarding the skills that must be possessed by information technologists also change. Typically, most of the problems that engineers encounter in practice differ from those they have encountered previously, and these problems are almost certain to be different from any that they have encountered at a university. For this reason, engineers must be able to apply concepts that they have learned during their education to the solution of real-world problems (Mills & Treagust, 2004). Consequently, software-engineering (SE) programs must be

designed to provide IT professionals with the necessary skills to manage this complex and unstable environment. However, it is well known that generally, SE professionals working in industry are unsatisfied with the level of real-world preparedness possessed by new university graduates entering the workforce (Callahan & Pedigo, 2002; Conn, 2002; McMillan & Rajaprabhakaran, 1999; Wohlin & Regnell, 1999).

## Problems of SE programs

Since the 1990s, technologists have expended a great deal of effort to make SE programs more up to date and more effective (Denning, 1992; Ford & Gibbs, 1996; Hartmanis & Lin, 1992;Hilburn & Towhidnejad, 2002; SE, 2004; Swebok, 2004). However, the field still suffers from many problems that are summarised in the succeeding discussions.

### *Rapid changes in technology*

SE is a young and rapidly changing area. Changes in computer and IT cause changes in both the kinds of systems that software engineers construct and the tools available for the construction itself. This dynamism creates the danger that academic institutions provide skills and knowledge that soon become obsolete (Ohlsson & Johansson, 1995).

### *Essential problems*

SE has its so-called essential problems. Brooks (1987) summarises these problems as managing complexity, maintaining consistency in changing systems and communicating difficulties. According to Pour, Griss and Lutz (2000), lifelong learning, group studies, industrial and academic cooperation, and self-regulation are some of the important considerations that SE programs need to cover in order to solve these problems. They conclude: 'Lifelong, self-directed education is also important. In a world of free agents and contractors, software engineers must pick up—on their own—many of their specialty skills' (Pour *et al*, 2000, p. 40).

### *Integrating knowledge*

Industrial organisations that work on software development face several problems. According to Lethbridge (2000), the most important issues for companies that are working on software development are software design and patterns, object-oriented concepts and technologies, requirements for gathering and analysis, analysis and design methods, human–computer interaction/user interfaces, databases, project management, configuration and release management, ethics and professionalism, presentations to audiences and leadership. The results of Lethbridge's study also show that these concepts are learned chiefly on the job—not at school—and that they are soon forgotten if learned during one's education. Therefore, even if an SE program's curriculum covers most of these concepts, learners tend to forget the concepts easily. One of the main reasons for this problem could be that in Software Engineering Education (SEE), these concepts are taught separately but that in a job-related experience, they are implemented in an integrated manner. Learners need to practise these concepts more often and more substantively in order to use them in an integrated manner.

**Challenges and opportunities of computer games**

Games or game-inspired exercises have been used in programming courses for a while (Adams, 1998; Baker, Navarro & Hoek, 2005; Becker, 2001; Bidarra, Dalen & Zwieten, 2003; Faltin, 1999; Feldgen & Clua, 2003; Feldman & Zelenski, 1996; Sindre, Line & Valvag, 2003). There are several reasons for using games in SEE. First, games have the potential to motivate learners. According to Sindre *et al*, such exercises are more fun, and thus, they motivate students more easily. Ferrari, Taylor and VanLehn (1999) also report that the fun element of a game makes the lessons learned more memorable. Because students are the end-users for games, they are able to compare different user-interface approaches among several games that they have experienced over the course of several years. This high degree of experience increases the quality of these programs in terms of usability (Sindre *et al*, 2003). In addition, games motivate learners to continuously work on and improve their skills. Students may continue to use and improve their products after the deadline, whereas most other exercise programs are delivered and then forgotten (Sindre *et al*, 2003). Visualisation is an essential tool in SEE, as it is used in the design, creation and interpretation of knowledge. Computer games enable students to encounter visualisation techniques and to puzzle formats in increasing levels of difficulty, and the games challenge students to develop a more advanced level of comprehension (Crown, 2001).

Games cover all kinds of topics that one may want to teach in SE curriculum (Jones, 2000). Several topics that are important for SEE are also applicable to the development of a computer-game project. These topics include computer graphics, AI, programming, SE, databases, human–computer interaction/user interfaces, physics, mathematics, operating systems, networks, object-oriented approaches, data structures and algorithms. Today, several universities offer courses and programmes about computer-game development (Bidarra *et al*, 2003; Feldgen & Clua, 2003; Kuffner, 2004; Paisley University, 2005; Shaw, 2001; Uclan, 2005; University of Luton, 2005).

The purpose of this study is to describe the implementation of a practice-oriented computer game-development course whose design should better prepare SE professionals to enter the workforce. The main assumption of this study is that a course that provides content geared towards developing widely applicable and adaptable skills should minimise the difficulties that stem from the transfer of learning to the workplace. In order to better understand the effect of this course on students' performance in a software-development project, I took 125 students (of whom 39 had taken the computer game-development course and the remainder had not) and investigated their performance in a 1-year senior-project course.

**Research methodology**

This study investigates the effect of a computer game-development course on students' performance insofar as a participant in a software-development project. The research question is as follows:

Does the computer game-development course improve students' performance in software-development projects?

*Data-collection process*

In this university, students need to fulfil the requirements attached to a senior-project course that centres on the development of an SE project. In order to answer the research question, I collected data that range over 3 years. I analysed the collected data to investigate the effect of the computer game-development course (Compe376) on the students' performance in a senior-project course (Compe492). Accordingly, I collected students' cumulative grade point average (CGPA), grades in Compe492 and grades in Compe376. Moreover, in order to get better feedback from the students about their opinions on Compe376, I conducted individual interviews with each student immediately after he or she had completed the course.

*Senior-project course (Compe492)*

Compe492 is a two-semester course for 4th-year students. The first part of the course requires students to conduct an independent research, which includes literature survey, problem formulation and the preparation of a detailed design for the solution of an SE problem. The students present the design in the form of project reports and seminars. For the second part, the students continue to work on the project that they began in the first part of the course, this time emphasising the development of the proposed system. At the end of the semester, the students must submit reports, present their activities and demonstrate the completed projects. A jury assesses the final product of the senior-project course.

*Computer games-development course (Compe376)*

The course is organised as a one-semester (13 weeks), three-credit technical elective course. The course comprises two main sections: practical studies and theoretical studies. A computer-game development environment (OpenGL and C++) serves as the main platform and guides student projects. Within this setting, students are asked to search for related information, to integrate new information with previously learned information and then to incorporate findings in their projects. They prepare reports, presentations and discussions to demonstrate what they have learned. The course objectives are fivefold:

- to learn how to design and develop computer games;
- to simulate the physics inside the computer games;
- to learn how to use two- and three-dimensional computer graphics and to create virtual reality;
- to learn how to apply concepts and techniques learned in other courses (such as AI, computer graphics, physics, mathematics, SE) to the computer games; and
- to improve object-oriented programming skills.

*Course content*

At the beginning of the semester, the instructor assigns a topic (eg, AI and physics in games, animations, graphics and so on) to each student, who studies it and presents it in the classroom. The instructor helps students to find information about the topics and to better understand and present each concept. During the first 7 weeks, the instructor

introduces main concepts concerning the game industry, game design and game development. After this period, each student presents his or her own topic during the rest of the course. Each student also prepares 10 questions and distributes the question to his/her classmates. The questions help students to be better prepared for the final exam, as these topics are all included in the final written exam. In the laboratory activities, the aim is to help students become familiar with the development environment (OpenGL and C++), which is chosen as a game development platform.

*Student evaluation*

In Compe376, the instructor assessed students' performance on the basis of four instruments: (1) students' work on the topic study (15%); (2) laboratory activities (20%); (3) term project (45%); and (4) final exam (20%). The instructor assigned four homework assignments in the laboratory activities. For the topic study, the instructor assigned students to a topic individually and required them to conduct a research study on that topic. The instructor asked them to present and discuss the results in the classroom. The final exam covered each topic presented by the students in the classroom as well as the concepts introduced by the instructor. For the term project, the instructor asked the students to design and develop a computer game. They were free to design any kind of computer game. There were only three constraints:

- The development platform should be OpenGL and C++.
- The size of the game should be more than 150 lines of code.
- Students were free to reuse library classes or code found on the net, as long as this reuse did not break any copyright regulations. They had to reference this code. Students were informed that violation of the copyright regulations would result in a failing grade.

The design part of the term project accounted for 15% of the course grade. The instructor evaluated the game-design reports according to the following factors:

- requirement-gathering process and documentation;
- software-development methodology;
- tools analysis;
- modelling tools such as Unified Modeling Language and Dataflow diagrams;
- object-oriented design techniques for the derivation and the use of design patterns (for example, the Model-View-Controller design for a game's user interface).

The development part of the term project accounted for 30% of the course grade. The instructor evaluated the development of the game project according to the following factors:

- practice in deriving object models from requirement analysis phases;
- quality-standard coding styles and documentation practices;
- quality control/assurance procedures;
- loosely coupled architectures, change-control techniques and performance tuning;
- usability and other user-interface issues;
- program functionality.

Finally, the students' presentation of their work and the demonstration of their games accounted for 10% of their total grade as a bonus. Also, the instructor asked the students in the class to rate their peers' presentations. The average of these evaluations, including the instructor's, constituted the oral presentation grade for the project.

*Students*

Students who were in their 3rd or 4th year in the department enrolled in the computer game-development course (Compe376). They also had to take the senior-project course (Compe492) during their last year. However, some students took Compe376 parallel to Compe492. Compe376 was offered to 39 students in the 2004 and 2006 spring semesters. From 2004 through 2006, 125 students took Compe492.

## Results

In the following sections, I present the results of this study to realise two objectives: first, to show the effect of Compe376 on students' performance in the senior-project course; second, to provide evidence of students' performance improvement in problem-solving skills, the application of previously learned knowledge, the use of independent learning and learning by doing.

*Quantitative data*

Table 1 summarises CGPA information of the two groups of students.

The average CGPA of the Compe376 students was slightly higher than that of the non-Compe376 students. I conducted an independent-sample *t*-test to determine whether or not the CGPA difference between these two groups is significant. It is not significant, $t\,(123) = -1.1$, $p = 0.23$. The 95% confidence interval for the difference in means is from –0.31 to 0.07. Table 2 summarises students' grades on the senior-project course.

*Table 1: CGPA results of students*

| *Compe376* | *N* | *Mean* | *Standard Deviation* |
|---|---|---|---|
| Not taken | 86 | 2.39 | 0.52 |
| Taken | 39 | 2.51 | 0.49 |

CGPA, cumulative grade point average.

*Table 2: Students' grades on the senior-project course*

| *Compe376* | *N* | *Mean* | *Standard Deviation* |
|---|---|---|---|
| Have not taken | 86 | 2.79 | 1.00 |
| Have taken | 39 | 3.17 | 0.93 |

I conducted an independent-sample *t*-test to evaluate the hypothesis that students who had successfully completed Compe376 would perform better in Compe492. The test was significant, $t\ (123) = -1.9$, $p = 0.049$. Students who had successfully finished Compe376 ($M = 3.17$, $SD = 0.93$), on average, were more successful in Compe492 than were the students who had not taken Compe376 ($M = 2.79$, $SD = 1.00$). The 95% confidence interval for the difference in means is from −0.7 to −0.001.

*Qualitative data*

Improvement in problem-solving abilities

Each student created a different story and game design for the term projects. Generally, they developed ball games, war games, space games, puzzles, labyrinth games, different Tetris implementations, board games and educational games. They also designed different objects such as destroyers, aircraft, tanks, factories, cities and a variety of characters. To generate their game stories, they used special simulations, 2-D (2-Dimensional) and 3-D (3-Dimensional) approaches, and tools such as MilkShape and OpenGL.

For the first 6 weeks, they searched for the techniques that they needed to use in their design. Some of these techniques were described during Compe376 course; but the students still needed to find out other techniques such as implementing AI in games, creating animations in games and using Frustum culling techniques. Students who needed these techniques to implement their game projects searched for information about these concepts, then they implemented these concepts into their games and also prepared lessons to show their classmates what they had learned. Once the students felt more comfortable within the development environment, they started to create some simple games. Towards the end of the semester, they started to develop new ideas about their games quickly. For example, one student reported,

> I spend most of my time [trying] to understand the structure of the environment that I was dealing with. But, after I solved it, I quickly managed to develop my project.

In general, students spent most of their time familiarising themselves with the development environment, acquiring an understanding of the game-development techniques and searching for the solutions that they needed to apply to game-related problems. For the implementation, they did not spend as much time as they had expected. For example, according to a student,

> I just sat in front of my computer and spent my whole night on the problem. In the morning, the skeleton of the program was ready. When I realized that I could create a new world and control it, I could not stop myself from spending more hours on the project.

Other students mentioned that once they had become familiar with the environment and had found several sources on the Internet, they easily and enjoyably developed their own games.

Applying previously learned knowledge

In this study, some evidence shows that the students managed to integrate previously learned information into their games. For example, one student implemented certain

physics laws therein to simulate force effects and acceleration. He said, 'These are very simple concepts that I learned in high school. But up to now, I haven't used these concepts in real life. In this project, I used and modified them to create my own world. It was very exciting'.

To develop their projects by using OpenGL and C++, the students further used what they had previously learned in their programming-language courses. Similarly, two students who had taken an AI course managed to implement some related techniques into their games. Six students who had taken the SE course before the game course got high grades (over 3 out of 4) on their game-design reports. This evidence shows that those students managed to implement their previously learned information into their game projects successfully.

Use of independent learning

At the beginning of the course, the students worried about their ability to develop a game. But after they understood the environment and learned how to find the information that they needed, they quickly managed to develop their own games. The students found most of the information on their own and then adapted it into their designs. They found some internet resources and shared them with their classmates. How they searched for information also reveals how they learned. For example, a student reported that,

> At the beginning of the course, I felt like I was lost. But, after some time, when I decided what to do, what information I needed, and where to look, everything became clearer to me. I learned a lot from the Internet sites. I got some source codes from these sites [which are provided as open source], and I adapted them to my project. Once I managed to learn by myself, creating my own game became easy. I have to say that the Internet is a great source for me to learn by myself. Everything that I needed was there; I just reached for information and worked on it.

Other students implemented several different concepts in their games. Most of these concepts were not discussed by the instructor during the class. For example, a 'Steel Pong' game developed by a student consists of two user-controlled bars, two horizontal static Table-boundary bars (when a steel disk hits these table-boundary bars, it bounces), and a steel disk that is capable of moving bidirectionally so as to interact with the player bars when it hits them. The student who created this game faced several problems during development. For example, he had to discover, on his own, techniques such as Frustum culling and then implement them in his game.

During the lecture hours, the instructor provided no specific information about SE processes. However, the instructor evaluated the game-design reports according to some of these factors such as the requirement-gathering process, the documentation of the requirements analyses, the methodology used during the software-development process and the tools used during the software-development process. The course results were encouraging. A total of 13 students who had not yet taken the SE course received high grades on their game-design reports. This result shows that these students searched for relevant information on their own in order to produce good-quality documentation.

Similarly, two students implemented some AI techniques such as search algorithms and path-finding algorithms into their game projects. These students had taken neither any AI course before nor a course parallel to Compe376. They searched for the related information both on their own and according to the requirements of their game design.

The instructor encouraged students to implement their topic-study research into their term projects. The results of the term projects show that 21 students managed to implement into their projects the ideas that the students had researched during the topic study.

Learning by doing

Students' performances in creating both the game designs and development projects revealed that all the students were ultimately successful in these aspects of the course. The students also managed to implement new ideas in their games. During the interviews, one student said,

> In this course, I understood that I learn best by doing. Once I decided on my project topic, I started to learn different concepts that I needed to implement in my project. I found the related information. I implemented it in my project. I [then] searched for [more information] and implemented [that] in my project. Even during the nights, I thought about how to implement a specific technique in my game. Even though I spent a lot of time on this course, it was fun. I learned a lot. Not from the instructor, but by myself.

Other students also reported that they learned more effectively by creating their own projects. Students in this course created their own worlds. To realise this task, they had to identify the most effective solutions to problems. In the process, they learned a great deal by implementing different solutions and by finding the most effective one. For example, a student said,

> At the beginning, I was faced with several problems that were hard to solve by myself. Afterward, I started to think about several different solutions. I implemented them, and accordingly, I managed to solve these problems.

## Conclusions

In Compe376, the instructor's experiences show that the motivation of students in this computer game-development course was very high. Students spent several hours applying new ideas to game designs and discovering the techniques that would make the realisation of the designs possible. Once the students had implemented a new technique, they continued to discover others. These findings support those of Sindre *et al* (2003), according to which both game projects in an introductory programming course and changes in course structure improved students' course performance in relation to students' course performance in previous years. Becker (2001) also found that the use of computer games as assigned programming applications in a 1st-year computer-science course helped students work harder and learn more.

My study's results show that students who had taken Compe376 performed better in the senior software development-project course than did the students who had not

taken Compe376. Improving students' skills in SE by means of a game-development course opens several challenges and opportunities for educators who try to prepare students for the SE field. First, a game-development course covers a wide range of concepts. Instructors should instruct students about matters concerning several technical and practical issues in game development. In turn, the students should quickly familiarise themselves with the development environment in order to design and develop their own projects. In each project, the process of deciding upon the game and the techniques to be used, and then searching for information about those techniques, could help students search for the proper solution and work individually. In general, the results of this study show that such project-oriented courses in SE programs can better prepare software engineers for industry demands.

## References

Adams, J. C. (1998). Chance-it: An object-oriented capstone project for CS-1. In *Proceeding 29th ACM Special Interest Group on Computer Science Education (SIGCSE) technical symposium on Computer science education*, Atlanta, Georgia, United States (pp. 10–14). New York: ACM Press.

Baker, A., Navarro, E. O. & Hoek, A. V. D. (2005). An experimental card game for teaching software engineering processes. *The Journal of Systems and Software*, *75*, 3–16.

Becker, K. (2001). Teaching with games: the Minesweeper and Asteroids experience. *Journal of Computing Sciences in Colleges*, *17*, 2, 22–32.

Bidarra, R., Dalen, R. V. & Zwieten, J. V. (2003). A computer graphics pioneer project on computer games. In *Proceeding Workshop Computaçao Grafica Multimedia Ensino-CGME, Workshop on Computer Graphics, Multimedia and Education*, October 8, Porto, Portugal (pp. 61–65).

Brooks, F. P. (1987). No silver bullet: essence and accidents of software engineering. *IEEE Computer*, *20*, 4, 10–19.

Callahan, D. & Pedigo, B. (2002). Educating experienced IT professionals by addressing industry's needs. *IEEE Software*, *19*, 5, 86–93.

Callahan, D. W. (2003). Development of an information engineering and management program. *IEEE Transactions on Education*, *46*, 1, 111–114.

Conn, R. (2002). Developing software engineers at the C-130J software factory. *IEEE Software*, *19*, 5, 25–29.

Crown, S. W. (2001). Improving visualization skills of engineering graphics students using dimple JavaScript web based games. *Journal of Engineering Education*, *90*, 347–354.

Denning, P. J. (1992). Educating a new engineer. *Communications of the ACM*, *35*, 12, 83–97.

Faltin, N. (1999). Designing courseware on algorithms for active learning with virtual board games. In *Proceeding 4th Annual Joint Conference Integrating Technology into Computer Science Education (SIGCSE/SIGCUE ITiCSE) conference on Innovation and technology in computer science education*, Cracow, Poland (pp. 135–138). New York: ACM Press.

Feldgen, M. & Clua, O. (2003). New motivations are required for freshman introductory programming. In *ASEE/IEEE Frontiers in Education Conference*, November 5–8, 1, T3C–T24.

Feldman, T. J. & Zelenski, J. D. (1996). The quest for excellence in designing CS1/CS2 assignments. In *Proc. 27th SIGCSE Symposium in Computer Science Education*, Philadelphia, USA, February 1996.

Ferrari, M., Taylor, R. & VanLehn, K. (1999). Adapting work simulations for schools. *The Journal of Educational Computing Research*, *21*, 1, 25–53.

Ford, G. & Gibbs, N. E. (1996). *A mature profession of software engineering*. CMU/SEI-96-TR-004, Carnegie Mellon University, Software Engineering Institute.

Hartmanis, J. & Lin, H. (1992). *Computing the future: a broader agenda for computer science and engineering*. Washington, DC: National Academy Press.

Hilburn, T. B. & Towhidnejad, M. (2002). *Software quality across the curriculum*. 32nd ASEE/IEEE Frontiers in Education Conference, Boston, USA.

Jones, R. M. (2000). Design and implementation of computer games: a capstone course for undergraduate computer science education. In *Proc. 31st SIGCSE Symposium in Computer Science Education,* Austin, TX, March 2000.

Kuffner, J. (2004). *15–493 Computer Game Programming Course, spring 2004*. Carnegie Mellon University. Retrieved 14 February 2007, from http://gamedev.cs.cmu.edu/spring2004/index.php

Lethbridge, T. C. (2000). What knowledge is important to a software professional? *IEEE Computer*, *33*, 5, 44–50.

McMillan, W. W. & Rajaprabhakaran, S. (1999). What leading practitioners say should be emphasized in students' software engineering projects. In H. Saiedian (Ed.), *Proceeding 12th conference on Software Engineering Education and Training* (pp. 177–185). Washington, DC: IEEE Computer Society.

Mills, J. E. & Treagust, D. F. (2004). Engineering education: is problem-based or project-based learning the answer? *Australasian Journal of Engineering Education*, 2003–04, online publication. Retrieved 14 February 2007, from http://www.aaee.com.au/journal/2003/mills_treagust03.pdf

Ohlsson, L. & Johansson, C. (1995). A practice driven approach to software engineering education. *IEEE Transactions on Education*, *38*, 3, 291–295.

Paisley University (2005). *Computer games technology*. Retrieved 14 February 2007, from http://www.paisley.ac.uk/courses/ug-courseinfo.asp?courseid=350

Pour, G., Griss, M. L. & Lutz, M. (2000). The push to make software engineering respectable. *IEEE Computer*, *33*, 5, 35–43.

SE (2004). *Software engineering 2004: curriculum guides for undergraduate degree programs in software engineering, August 2004*. IEEE Computer Society. Retrieved 14 February 2007, from http://sites.computer.org/ccse/SE2004Volume.pdf

Shaw, C. (2001). *CS 4455 video game design and programming*. Georgia Institute of Technology. Retrieved 14 February 2007, from http://www.cc.gatech.edu/classes/AY2003/cs4455_fall/

Sindre, G., Line, S. & Valvag, O. V. (2003). Positive experiences with an open project assignment in an introductory programming course. In *Proceeding 25th International Conference on Software Engineering (ICSE '03)*, Portland, Oregon (pp. 608–613). Washington, DC: IEEE, Computer Society.

Swebok. (2004). *Guide to software engineering body of knowledge*. IEEE Computer Society. Retrieved 14 February 2007, from http://www.swebok.org/

Uclan (2005). *University of Central Lancashire, computer games development and design*. Retrieved 14 February 2007, from http://www.uclan.ac.uk/courses/ug/subjects/compgamesdesign.htm

University of Luton (2005). *Computer games development BSc course*. Retrieved 14 February 2007, from http://www.luton.ac.uk/courses/bysubject/cominfsys/bsc-comgamdev

Wohlin, C. & Regnell, B. (1999). Achieving industrial relevance in software engineering education. In H. Saiedian (Ed.), *Proceeding Twelfth Conference on Software Engineering Education and Training* (pp. 16–25). New Orleans, LA: IEEE Computer Society.