Teams Battling Teams: Introducing Software Engineering Education in the First Year with ROBOCODE

1. Introduction

In order to be effective software engineers, computer science undergraduate students must master not only core theoretical computer science foundations and programming skills, but also a variety of competencies having to do with design, the preparation of software-related documentation, and soft skills relating to effective teamwork. In many traditionally structured computer science programs, the acquisition of these software engineering skills is localized in very few points in the already dense computer science curriculum: most commonly in an introductory software engineering course that precedes a final, senior year capstone course.

This curricular structure presents educators with two significant challenges: First, it makes the study and application of software engineering skills overly focused within the context of isolated, discrete courses. While understandably driven by the tight confines of the undergraduate curriculum, this isolation results in the perception that the content of such courses are a skill-set with limited applicability. Second, it means that students are generally unprepared for the challenges of software engineering learning activities when first encountered. With most introductory software engineering courses applying experiential learning and couching learning activities in the context of a team-based project, the challenge of mastering course content is complicated by what is, for most students, their first significant experience with teaming and the difficulties of managing not just their own work but also the work of their teammates.

These challenges motivate the need for better integration of software engineering knowledge into the computer science curriculum, and the earlier introduction of the related skill-set and team-based project experiences. In order to begin addressing these difficulties, we have introduced a challenging and engaging software engineering team project into our first year introductory programming sequence based on the ROBOCODE robotic combat simulator. Programming in the JAVA language, students work on developing a cooperative team of robots that competes in a tournament against robotic teams built by their fellow students – teams of students developing teams of robots. Our key goals with this curricular enhancement are: (a) to include software engineering education earlier in our program and in a more integrated manner than current practice, and (b) to introduce team- and project-based software engineering activities in a low risk, high student involvement setting in order to create a smoother learning curve for students. This paper contributes:

- A discussion of the learning theory foundations for our approach, based on experiential learning targeted at increasing student motivation;
- A minimally disruptive framework for better integrating software engineering education within a computer science curriculum by elaborating our course design plan, and providing a description of areas that required particular care; and,
- A presentation of quantitative and qualitative evaluation results, based on student surveys, evaluation based insights, and our own observations.

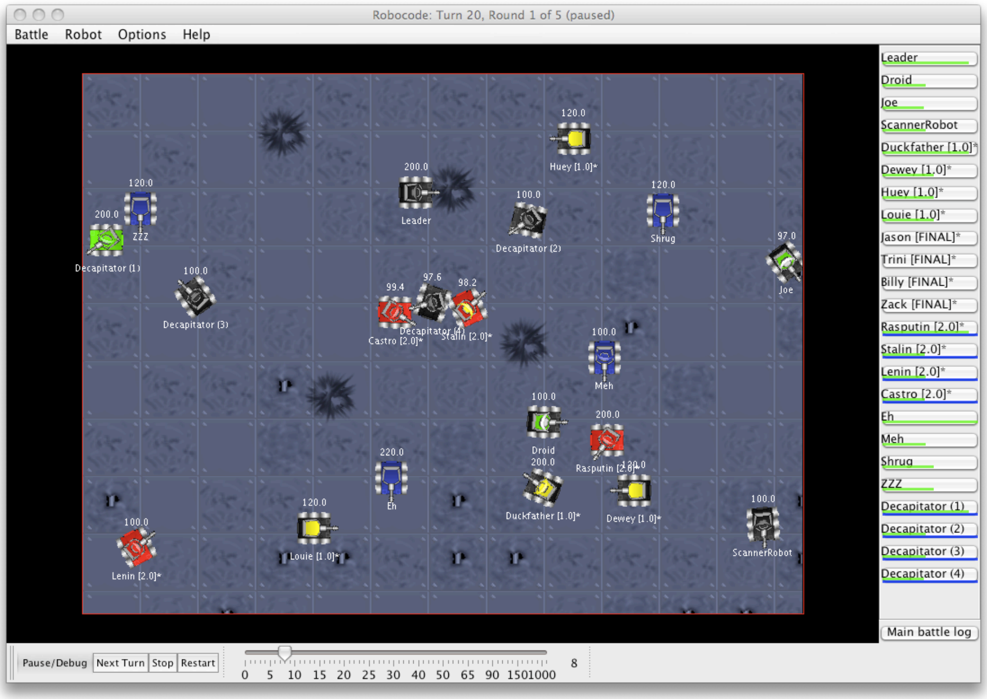

Figure 1. A screenshot of a ROBOCODE battle in progress.

The remainder of this paper is organized as follows: Section 2 discusses background information on learning theories and the Robocode simulator, Section 3 presents the design of our approach, and Section 4 discusses evaluative results while Section 5 offers concluding remarks.

2. Background

This section presents background information that underpins our work regarding the ROBOCODE simulator and the learning theory foundations that inform the design of our approach.

2.1. ROBOCODE

ROBOCODE is an open-source development platform for robotic combat simulation, intended to support the instruction of students in the JAVA programming language, originally developed and released to the public by IBM[8]. Students develop and program the logic for each autonomous robot, and these robots then compete against each other under the management of the simulator infrastructure; Figure 1 shows a ROBOCODE battle in progress, showing the graphical interface of the simulator, while a text-based view showing only final battle statistics is also available.

The basic objective of each robot is to survive while destroying its competitors: each simulated robot consists of a mobile platform that moves around the battlefield, a radar that collects information about other robots, and a turret-mounted gun that fires bullets. The key resource associated with each robot is *energy*, which is a measure of health since the robot becomes disabled when its energy is reduced to zero. Being struck by other robots' bullets, colliding with other robots, or colliding with the battlefield's walls causes a robot to lose energy. Firing bullets requires an investment of energy, which is regained only if a bullet actually strikes an enemy –

energy invested into bullets that miss is lost. The objective, then, is for students to program logic that allows robots to preserve their energy by avoiding collisions and enemy fire while also maximizing their own chances of striking an enemy. Developers use and extend core ROBOCODE classes, with the infrastructure providing information about events taking place on the battlefield and supporting an API that can be used to enact robot actions. The ROBOCODE framework also acts as a "sandbox" that restricts the actions that programs can execute and prevents one robot from accessing the functions of another.

While the ROBOCODE simulator has been previously used in introductory computer science courses[7], this previous work focuses on the acquisition of programming language skills through the development of a single robot by individual students. Our application is a novel divergence that leverages the simulator's capability to support the creation of robot teams, where robots of the same team may coordinate their actions at runtime, developed by teams of students and focusing on introducing software engineering skills, documents, and team-based activities.

2.2 Learning Theory Background

The foundation of our approach lies with the *learn-by-doing* learning philosophy[2], which espouses the importance of the interactive, social, and hands-on aspects of effective learning, rather than an exclusive focus on the material or information to be conveyed to the learner. The two expressions of this philosophy most relevant to our work are *situated learning*[6] and *problem-based learning*[9]. Situated learning identifies the importance of placing learning activities within a context that is fundamentally similar to that in which the skills and knowledge acquired will eventually be applied. Problem-based learning stresses activities that are open-ended and encourage self-directed learning, which fosters a greater degree of student investment and therefore quality of work.

Our work is also significantly informed by research into and findings related to student motivation. The ARCS motivational model[5] posits that there are four key factors in encouraging a high level of learner motivation: attention (fostering curiosity and self-driven problem-solving), relevance (supporting clear linkages of learning activities to learner interests), confidence (ensuring that learners maintain an expectation of success), and satisfaction (clearly showing learners the value of the learning outcomes being achieved). Further reinforcing student motivation is a high degree of learner involvement and commitment to the learning activities taking place, which are critical factors in fostering effective learning[1].

In our own work, we have strived to incorporate insights from these educational approaches and student motivational models. Our adoption of a team-based project strongly espouses situated learning, as most of the work of practicing software engineers is performed within the context of a team and not individually. Furthermore, students are free to design the behaviors of their ROBOCODE robot in whatever way they feel will make their robot most effective, which provides an open, problem-based context for their work. We have also found that the competitive setting of ROBOCODE tournaments is highly engaging, with students being energetically involved and invested into their robots' success, which directly supports the kind of student attention, satisfaction, and commitment to the project that supports effective learning.

3. Approach

Within conventionally structured computer science curricula, the inclusion of the software engineering body of knowledge meets with significant challenges. Driven by the tight confines of the curriculum, required by such factors as accreditation demands and the rapidly advancing pace of the field as a whole, the instruction of software engineering is understandably limited to few, discrete points in the curriculum. Generally, this results in students being unprepared for the rigor of software engineering activities when first encountered, particularly the difficulties associated with team-based projects that are an almost ubiquitous part of software engineering instruction[4].

In order to address these challenges, we first aim to provide for better initial exposure of students to software engineering and teaming through the inclusion of a team- and ROBOCODE-based development project within the context of our second semester introductory computer science course. With this curricular improvement, we aim to introduce software engineering expertise at a much earlier point in the computer science curriculum than is usual and to better prepare students for the rigor of team-based projects in a setting that is entertaining and lower-risk than the more realistic projects found in dedicated software engineering courses. While the subsequent sections provide details on a number of aspects of our approach, the table below captures the overall project timeline in order to provide context for subsequent discussion.

Course Timeline	**Activity**	**Related Deliverable**
Week 1-2	Elaborate software requirements and design	*Initial Report*
Week 2-6	Implement ROBOCODE robot team	*Robot Implementation*
Week 6	Discuss final design and divergence	*Summary Report*

3.1 Project Description

The fundamental aim for each student team is the development of a ROBOCODE team, consisting of four cooperative robots. The simulation framework provides facilities to support this type of development, primarily focusing on providing robots belonging to the same team with the capability to share information among them. This capability expands the possible functionality of robots to a great extent, since the arc of each individual robot's simulated radar limits the information it can gather about the battlefield. The ability to share data and action directives among teammates allows each individual robot to make decisions based on a significantly larger body of information and supports the development of cooperative behaviors. This allows for a wider set of strategies that can be employed: robot teams, for example, are able to coordinate their fire on the same opponent, or move so that they surround a target to prevent escape, or strategically position certain team members far from harm. The overall project is grounded in a multi-round tournament setting, where robot teams are pitted against each other until a single robot team emerges victorious.

From a student perspective, this capability introduces significant additional complexities and design challenges fundamentally stemming from the asynchronous nature of the simulated battle. One interesting issue becomes the validity of available information in the decision-making process: Robot team members may base their actions on information that is immediately available to them, but also on data collected by their teammates. As time goes on, however, this externally provided information is likely significantly less accurate, particularly as it pertains to the positions of enemy robots which are likely to have moved away from the position in which they were initially detected. Another challenging concern is adjusting the tactics used by a robot team based on updated battlefield information and potential setbacks: Consider, for example, a team of robots that coordinates their movement in order to encircle an enemy. If that enemy robot is destroyed while the coordinated movement is still in progress, this encircling team becomes vulnerable during their maneuver that now lacks its ultimate objective. Resolving and making design decisions on these kinds of coordination challenges tends to be a primary area in which students invest time while engaged in this project.

3.2 Curricular Integration

The software engineering centric ROBOCODE project is included in the final six weeks of the second semester of our first year introductory computer science sequence. The course focuses on an in-depth examination of object-oriented programming as well as an introductory coverage of topics such as data structures, recursion, algorithms and algorithmic complexity, and multithreaded programming – coverage of these topics is couched within activities performed using the JAVA programming language. In order to be minimally disruptive, the team-based ROBOCODE project forms a parallel thread of activities to homework and lab assignments more closely coupled with course content. For example, while the learning module concerning data structures is accompanied by a homework assignment and a lab assignment focusing on linked lists, there is no explicitly required ROBOCODE-related assignment for linked lists. However, since students have mastered this content, many elect to use lists, stacks, and queues as part of their robot team's logic for purposes such as managing potential targets. This is a common thread regarding the topics covered in the course: Students acquire additional programming knowledge and techniques, master them through traditional assignments such as homework and lab exercises, and then independently apply them in their ROBOCODE projects.

3.3 Group Formation, Management, and Evaluation

Formal cooperative groups, which are structured, stable, and involve relatively long-term commitments[11], are an important element of situated learning and a key component of our approach to improving software engineering education. Concerns such as group size, group membership, and day-to-day team interactions, however, are critical factors in either fostering success or being insurmountable obstacles to student activities[12], so cooperative groups must be organized and managed with care on the part of the instructor.

Based on insights for the effective formation and management of groups[10], we explicitly focused on addressing the following concerns in the design of our project:

- *Clarity of task explanation and objectives*. In addition to a detailed problem statement for the project and an outline of the programmatic details of robot team development, students were also allowed to gain insight into ROBOCODE through a preparatory individual tournament that preceded this initiation of this group project.

- *Group organization and leadership*. While we allowed students great leeway in the selection of their teammates, we also ensured that each team had at least one high-achieving student who seemed a good candidate for being team leader.
- *Monitoring and group dynamics*. Throughout the six-week period of the team project, instructional staff held informal meetings with each project team. Through these meetings and associated status updates, we worked toward ensuring that conflicts between group members were mediated as early as possible.

For effective group operation, it is also critical to provide evaluative feedback at both the group and individual levels[12]. For this project, each deliverable element was accompanied by a student-provided peer-evaluation that is used in a zero-sum grading methodology. This allowed us to assign individual grades that are a better representation of individual student contributions than the quality of final deliverables alone. As a result, individual grades reflect high or low levels of contribution by students, based on the evaluations of their teammates.

Each student is required to provide a peer evaluation along with project deliverables: this peer evaluation provides a scaling factor for the score of the submitting student and each of their team members. A scaling factor of 1 for a teammate would mean that, according to the submitting student's estimation, that teammate deserves to receive exactly the overall score is for the deliverable, a scaling factor of 0.9 would mean that they feel their teammate should receive only 90% of that grade, while a scaling factor of 1.1 would mean their teammate's contribution merits an extra 10%. As this is a zero-sum system, total reductions to the scaling factor for team members must be accompanied by equal increases to the scaling factors of other team members: the intent is to allow students to provide a measure of their teammates' contributions, with scaling factors smaller than 1 indicating that their teammate did not contribute as much as others and scaling factors greater than 1 indicating their impression that their teammate contributed more than expected.

The grade assigned to each deliverable by the instructor is the assignment's baseline score for each team member and the instructor then modifies this baseline score by the average of all the scaling factors assigned to each student. In addition to a tangible measure of student feedback about theirs and their teammates' individual scores, this also provides another indication to the instructor that an intervention is needed: any overall scaling factors of less than 0.9 or greater than 1.1 result in a group meeting with the instructor in order to discuss the group dynamics and inner workings that are contributing to such relatively large disparities of effort.

3.3 Software Engineering Lifecycle and Project Deliverables

In addition to providing the context for students to experience a meaningful team-based project, our goal is to expose students to important software engineering skills and artifacts, relating to project management, requirements, design, and reflection over the development process. While it is infeasible to expect students to follow a rigorous development process and produce complete associated documents and artifacts, we feel it is important for them to be introduced to these concepts through scaled-down versions of these activities and related deliverables.

Students participating in the ROBOCODE-based team project use a truncated waterfall lifecycle process and produce a number of software engineering documents and artifacts while engaging in the following activities:

- *Requirements and design*. Students are required to begin their software engineering activities by preparing an initial document that outlines the strategies and tactics their robot team will employ, along with a description of how they will design their robots in order to achieve these required behaviors.
- *Management plan*. Based on their requirements and design descriptions, students are further asked to create and document a project management plan that describes how they will go about organizing their development activities. They have to decide on and document who will lead their team, how they will organize their team meetings and activities, and how they will allocate work items to each team member.
- *Implementation*. At the end of the six-week development period, students submit the commented source code for each of their ROBOCODE team robots.
- *Summary and divergence report*. As the final deliverable of this team-based project, students are required to submit a final summary report of their project activities. This report outlines the final features of their robotic team and how they actually managed to achieve their desired goals. Perhaps most importantly, students are also required to address how their design and implementation diverged from their initial plans.

While students do not necessarily, and are not expected to, master formal software engineering techniques and methodologies, they gain valuable skills in addition to a significant team-based project experience. Through their work on requirements and design definitions, students gain experience with carefully identifying and documenting the features their system must exhibit before beginning their implementation, which is likely the first time in their careers as computer science students that they are required to do so. Through the elaboration of their team's management plan, students are forced to consider their own personal strengths and weaknesses as developers in order to reach effective decisions regarding work allocation. Finally, the requirement that they discuss how their final product diverged from their initial design necessitates that students carefully reflect over their development activities, the difficulties they encountered, and the validity of their initial estimations of the difficulty of project-related tasks. These are key software engineering skills that are normally not addressed until significantly later in the curriculum than the first year course that this ROBOCODE project is included in.

4. Evaluation Results

In our experience, this team-based ROBOCODE project has been effective and successful in engaging the interest of students as well as better preparing them for larger-scale, dedicated software engineering learning experiences. We base this assertion on a combination of informal anecdotal evidence and observations, as well as quantitative and qualitative data collected through student surveys administered over two semesters of offering this team-based activity.

4.1 Software Engineering Preparation

One of our fundamental goals with the introduction of this project is to inject software engineering learning into the introductory programming level. Our project is designed to require that students consider and document their decisions regarding requirements, design, and project management in addition to implementing their ROBOCODE team. As a result, the design of our project necessitates that students are exposed to these software engineering activities with explicit deliverables. Furthermore, they are also exposed to a team-based project setting, which is critical in building soft skills, such as effective and convivial communication, time management and scheduling with team members, and conflict resolution. The very fact that students are exposed to these learning activities in their first year meets our goal of introducing software engineering learning earlier in the curriculum.

Quantitative and qualitative survey data also support an increased level of exposure to and appreciation of software engineering skills by students. For example, students were asked to rate their level of interest, from 1 to 10, for various aspects of this ROBOCODE project that include problem analysis, building and applying programming skills, gaining the recognition of their peers, and winning tournaments. Of these categories, the "problem analysis and solving" category was rated the highest, with a median interest level of 8. One student reported that their favorite part of the assignment was "designing [their] modular classes and just thinking how the thing itself will come together" while another found the most useful element of the project to be "designing behaviors." To have junior students be this interested in design and problem analysis – both of which are core software engineering skills – over simply programming is a clear indicator of success in the context of our goals with this ROBOCODE project.

While we have not completed a structured, long-term longitudinal study of the effect of this experience on the grades of students in subsequent dedicated software engineering courses, we have collected data from two software engineering course offerings that were staggered around the ROBOCODE projects described here: the first software engineering course was offered to students that had not participated in the ROBOCODE activity, while the second one consisted of students that had. While we were not able to strictly control student membership for previous academic performance, we offer the following metrics as early informal indicators: In the software engineering course that followed ROBOCODE projects, the overall median grade was roughly 1% higher (an insignificant difference) but the overall median grade on the requirements elicitation and documentation assignment was 7% higher. While we do not notice significant differences in overall performance, we hypothesize that the relatively significant performance differences in the requirements-centric assignment may be due to better preparation on the part of the students for requirements elicitation and documentation activities.

4.2 Student Involvement

We are very confident that this project fosters a high degree of student enjoyment and involvement. Perhaps the best indicator of enjoyment is the degree of enthusiasm exhibited by students during in-class ROBOCODE tournaments. During tournaments, students are quite active in cheering their robots on, congratulating each other on hard-won victories, and openly expressing disappointment upon losses. We posit that the competitive tournament setting of the project underlies this enthusiasm and involvement, which we are quite pleased to note exceeds that which we have experienced using other kinds of projects that did not involve competition, such as media-based assignments[3]. Involvement into the project was also evident in the amount of time and effort students invested into their work, which was significantly larger than that invested into the more traditional assignments of the course.

These assertions are also based on quantitative data drawn from surveys deployed to two cohorts of students participating in this ROBOCODE-centric project:

- Students were asked to rate their enjoyment of the Robocode team-based assignment, and 76% of students said they either “enjoyed” or “greatly enjoyed” the assignment. Only 14% of students rated their experience at some degree of lack of enjoyment.
- Students were asked to report on how many lines of code (LOC) they wrote for their Robocode implementations, and 41% of students said they each wrote over 1000 LOC, while 33% reported that they wrote well over 1500 LOC. This tangibly demonstrates student investment: compare the size of programs to the single largest conventional, non-ROBOCODE assignment for the course measures that measures roughly 300 LOC.
- Students were asked whether the ROBOCODE assignment made them more enthusiastic about their work in this course, and 75% of students either “agreed” or “strongly agreed” with this assertion.
- Students were asked if they enjoyed the competitive nature of the Robocode assignment, and 90% of students either “agreed” or “strongly agreed” with this statement.
- Students were asked whether they preferred this ROBOCODE assignment to the other, more conventional assignments of the course: 67% of students either “agreed” or “strongly agreed” that ROBOCODE provided a more preferable experience.

Qualitative data also supports our conclusions regarding an increased level of student involvement, commitment, and enjoyment. One student commented that the assignment was “a creative way to code while the tournaments inspired programmers to go that extra mile” while a second commented that the assignment was “new and different from a traditional assignment.” A student also said that the project is a “cool idea, Robocode is addictive” and another student reported that their favorite part was “watching the battles, even though I got last almost every time.” A favorite quote from a student was “I feel I learned much more about programming in this than any other part of the course.”

4.3 Negative Perceptions

While we are very encouraged by the reception of this project structure by students, the response was not universally positive. The negative comments that students made primarily revolved around two fundamental issues: the time-consuming nature of the team project, and technical issues with the ROBOCODE simulator. One student, for example, says “in order to do well, a much greater proportion of outside study time was needed as compared to a typical 3 credit class.” Another states that the project “added to an already high workload, more stress.” These comments are understandable, as the team-based nature of the project adds a significant amount of overhead in terms of the time that students need to invest into the project in order to do well. For future offerings of the course, we intend to provide more time for students by increasing the 6-week time period allocated toward the team ROBOCODE project, in the hopes that the additional time without a corresponding increase in the project's requirements will provide a more forgiving schedule for students. Another significant issue, particularly for certain students, was experiencing technical issues with the ROBOCODE simulator. While most students did not have undue difficulty, some had to work at circumventing significant simulator bugs, such as memory leaks. We anticipate that future versions of the simulator will resolve these technical issues.

5. Conclusion

Within the structure of conventional computer science curricula, software engineering learning is isolated in few discrete courses, which makes it challenging to properly prepare students for the rigor and challenges of the experience. In this paper, we describe our initial effort in addressing this challenge by introducing software engineering learning activities in the form of a team-based development project centered on the ROBOCODE simulator that we introduced into our first year programming course sequence. In this project, teams of students develop a team of ROBOCODE robots that compete against the teams of their classmates, and students engage in software engineering activities such as requirements gathering and design, produce software engineering deliverable documents, and experience a significant team-centric development experience. Our own observations along with qualitative and quantitative survey data supports that our learning-theory based design is effective in stimulating student interest and providing students with a gentle introduction to software engineering learning.

6. Acknowledgements

The author would like to thank Eck Doerry for developing the basis of the zero-sum peer evaluation system we adopt, the many open-source contributors and developers of ROBOCODE, and the computer science students of Northern Arizona University. This work sponsored in part by the National Science Foundation under NSF Grant CCF-1017408, and Northern Arizona University's Faculty Grant Program.

Bibliography

[1] Astin, A.W. 1993. What Matters in College? Four Critical Years Revisited. Jossey-Bass.

[2] Dewey, J., Democracy and Education: An Introduction to the Philosophy of Education, The Macmillan Company: New York, USA, 1916.

[3] Forte, A. and Guzdial, M., Computers for Communication, Not Calculation: Media as a Motivation and Context for Learning, in *Proceedings of the Hawai'i International Conference on System Sciences*, Big Island, Hawaii, 2004.

[4] Hayes, J.H., Energizing Software Engineering Education through Real-World Projects as Experimental Studies, in *Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET)*, IEEE Computer Society, Covington, KY, USA, 2002, pp. 192-206.

[5] Keller, J.M. and Suzuki, K., Use of the ARCS Motivation Model in Courseware Design, in *Instructional Designs for Microcomputer Courseware*, Jonassen, D.H. (Ed), Lawrence Erlbaum: Hillsdale, NJ, USA, 1988.

[6] Lave, J., Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life, Cambridge University Press: Cambridge, UK, 1988.

[7] O'Kelly, J. and Gibson, J.P. 2006. RoboCode and problem-based learning: a non-prescriptive approach to teaching. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pg. 217-221.

[8] http://robocode.sourceforge.net

[9] Savery, J.R and Duffy, T.M., Problem Based Learning: An Instructional Model and its Constructivist Framework, in *Constructivist Learning Environments: Case Studies in Instructional Design*, Wilson, B. (Ed), 1996, pp. 135-148.

[10] Smith, K.A., Cooperative learning groups, in *Strategies for Active Teaching and Learning in University Classrooms*, Schomberg, S.F. (Ed), Continuing Education and Extension, University of Minnesota, Minneapolis, MN, 1986.

[11] Wankat, P.C. and Oreovicz, F.S., Teaching Engineering, Knovel, USA, 2006.

[12] Wilde, D.J., Using Student Preferences to Guide Design Team Composition, in Proceedings of ASME Design Engineering Technical Conferences, DETC97/DTM-3890, Sacramento, CA, USA, 1997.