Enhancing Engineering Education with Writing-to-learn and Cooperative Learning: Experiences from a Software Engineering Course

Lonnie R. Welch, Sherrie Gradin, and Karin Sandell
Ohio University
Athens, OH 45701
welch/gradin/sandell@ohio.edu

1. Introduction

Current progressive teaching movements draw forth strong skepticism as they often seem antithetical to engineering classes. Why would anyone want to switch from the lecture method of teaching engineering to methods that employ active learning? Doesn't lecturing produce the most informed engineers? Isn't lecturing the best way to challenge students? To uphold the highest standards? Many hold the view that active learning methods may be appropriate for "soft" disciplines, but are inappropriate for engineering and the sciences. Others argue that students won't take the course work seriously and that coverage of material would have to be sacrificed. The presenters will question the validity of these objections by defining learning goals, such as depth of learning, engagement, and retention, that should be considered during selection of teaching methods.

It will be shown that teaching writing-to-learn and cooperative learning achieve these goals and result in extraordinary transformation of both teacher and students. Student engagement and excitement are elevated at the same time as the depth of learning increases. Students become better engineers because they can think critically, solve problems individually or in teams, write better, and orally present information. Teachers find themselves challenging students with an even more demanding curriculum. Examples from a software engineering course will illustrate how these methods can challenge students more, create higher standards for learning, and produce better engineers than a typical lecture approach to teaching.

2. Organization and Goals of the Software Engineering Course

This paper describes how cooperative learning and writing-to-learn have been employed in Software Design (course CS 456[13] in the School of EECS at Ohio University). The purpose of the course is to provide students with skills needed in the software engineering profession. While they have completed numerous courses requiring development of software (students take CS 456 in their senior year), they typically lack several important perspectives. They have focused

almost exclusively on the *implementation phase* of software development, which is only one step in the modern software development processes employed by software engineering professionals. Thus, the skills taught in this course involve all the phases of the popular Unified Software Development Process[1].

Another goal of the course is to provide students with knowledge and experience relevant to (1) working in a software engineering team and (2) interacting with a customer. The course covers material about interpersonal communications[2], both with members of a team and with a customer; students apply the material by working in teams to perform software engineering tasks for a customer.

In summary, students who successfully complete the course learn how to

- perform all major phases of the software engineering lifecycle (requirements, analysis, design, implementation and testing),
- use the Unified Modeling Language (UML),
- employ the Unified Software Development Process,
- perform unit, integration, and system testing,
- effectively participate in software engineering teams,
- lead software engineering teams,
- resolve conflicts,
- interact successfully with customers, and
- make formal presentations of software engineering products.

This is not a lecture-based course. The Professor is not a "sage on the stage," but is a "guide on the side." Student learning in the course is typically very high, but students must take responsibility for their own learning. The teaching methods that are used to facilitate learning include (1) application of course material to case study problems, (2) writing-to-learn, (3) discussion, (4) problem-based learning, and (5) cooperative learning. While these techniques result in deep learning and increased knowledge retention, they do require students to be prepared, present and engaged in all class meetings. This paper does not address all five of these techniques, but focuses primarily on writing-to-learn and cooperative learning.

3. Cooperative Learning

Chickering and Gamson[6], in a meta-analysis of research studies examining variables linked to student learning outcomes, located seven principles for good practice in undergraduate teaching, including "good practice encourages cooperation among students." Working together in teams, students become more involved in their learning and strive to assist each other in attaining course goals. As students listen to each other and try to connect their peers' interpretations to their own, they improve their listening skills and their abilities to evaluate different points of view.

The formal application of cooperation in the classroom, referred to as cooperative learning (CL), is founded in the social psychological theories of Morton Deutsch and Karl Lewin[4]. Similar in theory to collaborative learning, CL can be considered a subset that is more

highly structured and focused on a specific outcome, such as learning to master a procedure. The social theory supporting CL identifies several positive outcomes associated with working closely in a supportive group of individuals. Learning something new often poses a certain amount of risk and the social support of fellow team members provides the environment in which risk can be managed productively. Involvement and motivation are both critical to the learning process and both occur with highly functioning cooperative teams. Learning remains a social activity and CL groups emerge as communities of students where social skills emerge hand-in-hand with specific course learning goals[5].

Research examining student achievement connected with CL has found increased learning and increased satisfaction in CL classrooms[4, 5]. Solitary learning becomes a stressful, competitive process, while working with others cooperatively becomes more satisfying and stimulating. Students become committed to their fellow team members and work hard to accomplish their mutual goals and not let each other down. Explaining their positions to each other, CL team members gain confidence in what they know and are more likely to retain their new knowledge.

The use of CL teams has caught on in many college classrooms for an additional reason. Much of the workplace, including the workplace for software engineers, has become a team environment. Team experiences in the classroom become additional training and preparation for successful transitions to the workplace. Team building skills are a natural outgrowth of CL in the classroom and students thus enter the work force with the ability to contribute successfully in a number of different work settings.

The word "functional" appears in connection with "cooperative team" in the literature exploring this pedagogical strategy and remains the key to successful applications of CL in the classroom. The extensive reliance on group work and projects in some disciplines has led to student dissatisfaction because a lack of planning by the instructor has resulted in "dysfunctional" teams or groups. For example, group grades often are emphasized and the pressure of grading often breaks down the CL atmosphere. Students simply assigned a mutual task without supervision and without being accountable for their individual work may succeed, but oftentimes fail as the burden of the task falls on the shoulders of one or two highly grade conscious individuals. Group members shift their focus from learning to the graded outcome and begin to respond as individuals concerned about their own graded outcome rather than as members of a community concerned about each other's learning.

The goal of effective CL becomes creating and maintaining highly functional teams that tackle complex problems with great energy and resources, working steadfastly until all their members have successfully concluded their task. These groups work in the classroom; the instructor turns over a certain amount of class material to the teams for their work and they learn together as they work through a task, rather than sit passively and listen to a lecture. The preparation ahead of time is considerable, as the instructor needs to carefully construct the exercise to introduce the students to the targeted material. The required outcome needs to be truly mutual and each student on every team needs to be individually accountable in some way for this outcome. Finally, the instructor needs to monitor and intervene in the groups' work, to

make certain that the teams know how to work effectively. Similarly, the student team members need to reflect on the success of their teamwork and prepare to improve their work on future exercises[4].

Many models exist for using CL in the classroom. In the jigsaw[3], the instructional material for a class session is divided up into a number of parts. Groups of students receive one of the parts and work together to prepare to teach this part to other groups of students. In the final step, teams are formed with someone representing each part of the material and the students on the teams take turns teaching each other their parts. The group concludes by summarizing all of the material from the session. Other CL models build competition among teams in the classroom, perhaps assigning a particular software engineering problem for the class session and awarding some bonus to the team of students that successfully solves the problem first or that comes up with the best solution. Team members work together and as they conclude their work any team member may be called upon to explain the outcome, thus ensuring that they have all taken part in the exercise.

Working together, students can learn a great deal, both about the subject matter at hand, as well as about the positive outcomes associated with cooperation and collaboration. As the demands of the curriculum grow, instructors need to explore ways to effectively engage students, enhance their critical abilities and assist them in assimilating an increasingly complex body of knowledge. Just as teams of software engineers are assembled in the workplace to manage the demands for high quality outcomes, teams of students can be assembled in the classroom to meet the need for high quality learning.

The remainder of this section presents examples of cooperative learning from the software engineering class.

Cooperative Learning Activity 1: Software Engineering Team Project

One of the fundamental concepts taught in this course is a software engineering process. The reason for this is that even moderately mature software development organizations employ a documented, standardized process. In addition to applying a rigorous software process, a software engineer needs to function productively in a team. To obtain experience in a team project, teams of students apply the software engineering process and group communication skills taught in the course to solve a real problem. Students also learn how to make a formal presentation to a customer and have the opportunity to practice this skill through presentation of the project artifacts.

The team project lasts the entire term and provides a *problem* that helps to create a desire in the students for learning. Each class session allows students to learn a new software engineering concept and to work with team members to apply the concept to their project. The team project is used in a variety of cooperative learning activities, including the following:

- mini-lecture with immediate in-class application,
- peer teaching, and
- small group discussion.

Cooperative Learning Activity 2: Mini-lecture with Immediate In-class Application

With this technique, new material is presented in a mini-lecture. A concept is taught and an example is given; students spend the subsequent portion of the class session working with their software engineering teams to apply the material to their projects. For example, a lecture about white box testing could be followed by an activity in which students, having completed the implementation of software artifacts, develop white box tests for the artifacts.

When the transition from lecture format to an active learning format was initiated in CS 456, it was noticed that when the students begin to apply the material in class they would "wake up" and start asking questions about the material; this caused the realization that the students often were not engaged during lectures (and thus did not learn much from the lectures) but that they are engaged and learning when trying to use new material to solve problems.

Cooperative Learning Activity 3: Peer Teaching

The peer teaching activity, a variation of the jigsaw, has worked very well. In this activity, software teams are divided into 2 sub-teams and each sub-team is given 10-15 minutes to prepare to teach material to the other sub-team (each sub-team is given different material to prepare). Following the preparation time, each sub-team spends 5-10 minutes teaching the material to the other sub-team. Finally, teams apply what they've learned to their team project or to an interpersonal communication scenario.

For example, this method could be used to teach system testing. Students would be asked to sit with their software engineering team members. Half of the students in each team would be asked to prepare to teach the concepts of *installation testing* and *configuration testing* to the remainder of their team; the remaining members of each team would be asked to prepare to teach the concepts of *negative testing* and *stress testing*. The sub-teams would teach the concepts to each other and then the teams would spend time in class producing one of each type of test case (installation, configuration, negative and stress). During this activity, the instructor would move around the classroom, monitoring the groups, and, if needed, helping them to understand how to perform the assigned tasks.

It is exciting to watch the students discussing the course material among themselves during this time. The level of student engagement with the course material is high; furthermore, it is rewarding to observe the students becoming immersed in course material on which I have not even lectured. That the depth of learning and retention with this method are superior to what can be achieved with the lecture method is self-evident when one observes this method in operation.

Cooperative Learning Activity 4: Small Group Discussion

Another way that new material is learned in CS 456 is by having groups of 2-3 students discuss how the material applies to a particular situation or problem. Below is an example that was used to help the students to learn the stages of group development.

> Interview an individual who is **NOT** a member of your software engineering team to obtain answers to the following questions. Write the answers in the spaces provided. After you have completed the interview, switch roles and let that person interview you.
>
> 1. What is the current stage of development of your software engineering team? Why do say that? (See pages 246-249 for descriptions of the stages.)
>
> 2. Name the stages of development that you have observed in your team. What are the specific behaviors that you observed in each of the stages?

This activity worked very well. It was not necessary to lecture on the stages of group development. The students were interested in knowing the stages of development traversed by their groups and thus eagerly learned the material from the textbook. The act of discussing the stages with another provided a cooperative, supportive setting for learning new material; the discussion also helped students to learn the material more deeply, because of the need to articulate understanding to another person. Applying the material to their own group situations also helped students to learn.

Cooperative learning has been effective in the software engineering course. Additionally, the reflective, individual approach of writing-to-learn has improved student engagement and learning in the course.

4. Writing-to-learn (WTL)

Writing-across- the curriculum (WAC) programs arose in the United States in the 1970s as a response to longstanding complaints about students' writing abilities and to the call for progressive education[7, 9]. Two primary components of most WAC programs are working with faculty to understand writing as a specialized discourse in a discipline or field, and writing as a tool for learning and mastering subject matter. We focus this paper on writing-to-learn rather than specialized discourse.

As a theoretical model, writing-to-learn posits that informal and process approaches to writing engage students in cognitive activities that foster critical thinking and deep learning. Scholars in fields as varied as rhetoric, psychology, and language theory have argued that writing is itself a cognitive and thinking process that when enacted is "connective," "selective," "active, engaged, personal," and "integrative in perhaps the most basic possible sense: the organic, the functional . . . [involving] the fullest possible functioning of the brain . . ." [8]. Practical applications of writing-to-learn theory suggest important changes in how we deliver our curricula, imagine acts of student learning, and assign writing.

Writing-to-learn activities and strategies differ substantially in means and ends from traditional, formal writing. The following comparison highlights a few of these differences.

TRADITIONAL ASSIGNMENTS:	**WRITING-TO-LEARN ACTIVITIES**:
Assigned as homework (often a relatively lengthy paper or report)	Assigned impromptu, often completed in class, may also be homework, often short (less than a page)
Process - → Product (student's intellectual work finished when the product is turned in)	Process → More Process (writing=thinking=more thought)
Graded on A/B/C/D/F basis by teacher (i.e., heavy investment of teacher's time)	Usually ungraded, but credit given or not given based on clear criteria (i.e., less formal grading by teachers)
Writing to test (is student writing/thinking right or is it wrong?)	Writing to think (Intellectual engagement is goal; error is a natural part of learning)
Asks students to be sure about what they write ("what's your thesis?")	Allows students to voice and explore questions
Students see writing assignments as penalty situations (no one ever gets 100% on this test)	Students see writing as a tool, a way to help them think about new material

If writing IS thinking then constructing writing-to-learn activities serves our students' learning by asking them to push beyond a surface understanding, by asking them to engage in a process of knowledge making, by asking them to think through a variety of perspectives, theories, or ideas, and by taking at least some responsibility for their own learning. The strategies for inviting students into this learning through writing to learn are numerous. Project logs, dialogue journals, memo exchanges, dialectical notebooks, guided writing, exploratory writing, and microthemes are just a few examples. Importantly, many of these strategies can be integrated into an already established curriculum without great disruption or terrific burden placed on the teacher. Writing-to-learn activities can be as long or short as teachers need them to be, placed

throughout the course or at selective key moments, read and responded to at length or not at all by the teacher, graduate assistants, or other class members.

Examples of writing-to-learn activities used in the software engineering course are provided in the remainder of this section. It is important to note that these are different from short-answer questions or quizzes. WTL questions are intended to be exploratory and to help students to examine the possibilities. WTL is informal, so the point is not to offer a tidy, singular answer, but rather to explore what, why, and how something might be answered.

Writing-To-Learn Activity 1: Guided Writing

New concepts are sometimes introduced to students through in-class writing-to-learn activities. Instead of lecturing on a topic, students are asked to read short segments of an article or of the textbook and to answer questions regarding the reading. The writing activities are typically followed by discussion. This method of teaching is superior to the lecture method, because *every* student is involved in the learning process and the level of engagement is deeper than when facts are presented by a teacher. This section presents 2 examples of this form of writing-to-learn. The first example involves learning software unit testing methods and the second example involves learning how to prepare a speech goal for a software engineering presentation.

EXAMPLE 1: **Unit Testing Activity**

```
The following questions refer to the article "Bridging the gap
between black box and white box testing," by Brain Bryson.

Read page 1 of the article (and read the first two sentences at
the top of page 2) and then follow the instructions below.

According to the paragraph at the bottom of page 1, what are
"black box testing" and "white box testing?"

The last paragraph on page 1 illustrates the concepts of black
box (BB) and white box (WB) testing for a soda machine. Write
your thoughts about what BB and WB testing would involve for
software.

Read the first complete paragraph at the top of page 2 and then
perform the following steps.

You will now perform a series of steps that will lead you through
the development of BB tests for one of the components for your
team project.

1. Select a software component that you have already implemented.
What is the name of the component?
```

2. Briefly describe the specification (set of requirements) for the component.

3. Define a series of inputs that can be used to perform BB testing of the component. For each input, indicate the expected output.

4. Is the test suite sufficient? Would you assume liability for the software after running your tests? Why or why not?

Read the second complete paragraph on page 2 of the article and then perform the steps below.

You will now perform a series of steps that will lead you through the development of WB tests for the components for which you developed BB tests.

1. How many execution paths are there through the code of the component? (Note: each if-statement results in two or more paths.)

2. What percentage of the paths would be covered by performing your BB tests?

3. Produce a series of inputs that will exercise every line of code in the component. For each input, indicate the expected output.

4. Would it be safe to say that the component is bug-free after you have performed your BB and WB tests? Explain.

EXAMPLE 2: SPEECH GOAL ANALYSIS

Each team uses powerpoint to make presentations of the software products that it produces. Each student must make part of each presentation. To teach them how to prepare a speech/presentation, students are given a set of questions, which guide them through the process of preparing a speech; they learn the concepts by *doing* them in class. This is an effective method because the questions posed motivate them to learn the material. The specific questions used to teach students how to write a speech goal are:

1. According to Communicate![2], an important step in preparing a formal presentation is determining the goal. The general goal is the intent of the speech, which can be to entertain, to inform or to persuade. What is the general goal for the presentation that you will make in the next class?

2. The specific goal is a single statement that specifies the exact response the speaker wants from the audience. Write the specific goal for the presentation that you will make in the next class. Write at least three different versions of the goal. Make sure the goal contains only one idea and that it indicates the specific audience reaction desired. (See fig. 12.5 of Communicate! for examples of specific goals.)

3. Read the accompanying speech and then answer the following questions. (Use the back of this sheet if necessary.)

a. What were the speaker's general goal and specific goal?
b. Was the goal clearly stated in the introduction?
c. Was the goal implied but clear?
d. Was it unclear?
e. How can this analysis help you to clarify your own speech goal?

Writing-To-Learn Activity 2: Toulmin Analysis of an Article

Students perform Toulmin Analysis of a journal article about software engineering. While this activity teaches students important state-of-the-art concepts about software engineering, it also introduces them to an effective critical thinking strategy and exposes them to graduate research. The following is the in-class activity that is performed:

Analysis of Research Articles using Toulmin Logic: To prepare for scholarly work leading to a thesis and/or to conference and journal article publications, graduate students will read articles related to their research topic. For each article, students will analyze the soundness of the authors' arguments by employing Toulmin logic. Toulmin logic is used by identifying the following:

- Claim(s)
- Grounds
- Warrant
- Backing
- Rebuttals

A *claim* is a hypothesis, conclusion or proposition made in the article.

The *grounds* support the claim; these may include proofs, quotations of experts, analyses, or experimental results.

The *warrant* provides a justification for the grounds and shows their relevance to the claim.

> Support for the warrant is referred to as *backing*.
>
> *Rebuttals* are counter arguments, counter examples, conflicting information, flaws, and other reasons for not accepting the claim, grounds, warrant and/or backing.
>
> Read the accompanying article ("A usage-model-based approach to test therac-25," by P. Hsia and others, 1995) and identify the claim(s), grounds for each claim, warrant and backing. Produce your own rebuttals. In addition, if the author has presented rebuttals, identify those.

Writing-To-Learn Activity 3: Minute Paper

A minute paper is sometimes used to assess student learning. At the end of a class session, students are asked to summarize what they learned. To limit the lengths of their responses and to cause them to write focused responses, they are usually asked to write their responses on 3x5 index cards. This has resulted in valuable feedback about the areas where student understanding is inadequate. Below is a minute paper assignment from a recent class:

> In the space below write an informal "note" summarizing what you understand about the entire analysis workflow. Explain where you had difficulties, questions and/or issues.

Feedback received from this prompt included "I am having difficulty with the organization of the textbook," and "I don't understand how what we've learned fits into the overall software engineering process." This feedback was helpful, because it provided a realization that the students were struggling with contextual issues. In response, email messages were sent to the entire class, providing guidance about how to use the textbook and explaining the overall software engineering process.

5. Summary and Conclusions

When one observes Professor Welch's software engineering course it is apparent that the course is fundamentally different from the traditional lecture course. At first glance the classroom looks chaotic. Chairs are not in straight rows. Professor Welch is not at the front of the classroom delivering the day's course material. Students are not quiet. Instead, students are huddled in groups, talking, pointing out passages in the text to each other, and writing. Professor Welch moves around the room, stopping to listen to groups of students and offering information and explaining concepts if necessary. What might appear chaotic is in reality students at work in an engaged, active manner. Activities change throughout the period from exercises where the students teach each other a concept, to taking individual quizzes followed by group quizzes on the same material (where students discuss and argue about why they answered the quiz questions in the manner in which they did). Why do this? As an outside observer it seems that students

really have to learn the material because they have to defend their answers and explain why they think one answer is better than another.

But are students learning more than in the traditional lecture classroom? The answer is not without complexity. When asked this question several students were not sure. There were things they liked about the lecture style of teaching. And yet, they all spoke about Professor Welch's alternative approach as a class where they could not get away with not reading, with not taking responsibility for learning the material. They all recognized that they were now better prepared to face conflict and to explain their ideas and concepts to others. Some felt their formal writing had not gotten significantly better, and yet, most of them recognized that the writing-to-learn activities helped them understand the course material more fully. Two Asian students commented that the interactive group work and problem solving as well as the writing helped them hone their English communication skills—both oral and written. They were both aware that they would be working in a global community in which English was the primary language of idea exchange. One young woman, an advanced undergraduate in her fifth year, argued eloquently that this course has taught her more in a few weeks than all of her lecture courses had in four years. She explained that she understood concepts fully, engaged in the material deeply, now had an understanding of how to work in a team context, and finally, that she could face problems and contexts with a positive end result.

In summary, this paper discussed cooperative learning and writing-to-learn in general and showed how these methods can be used in software engineering education. The results reported in this paper are regarding the use of the techniques in a software engineering course over the past two years. During this time its has been observed that

- student learning improves,
- engagement (of both students and teacher) increases,
- depth of learning increases,
- students enjoy in-class activities more than listening to lectures, and
- students have to work harder and are responsible for their own learning.

However, there also can be negative aspects of these approaches. Students may resent having to take responsibility for their learning. Furthermore, students who are in early stages of intellectual and ethical development[11, 12] may be scared by the new context, wherein the professor is not a guru who imparts knowledge and wisdom for the entire class session, and students must sort through information and draw their own conclusions.

We have found that negative effects can be lessened in several ways. Adopt new teaching methods gradually. Early use of cooperative learning and writing-to-learn methods in the software engineering class were not well received by the students, in part because too many changes were made too quickly. Do not get discouraged if you see many negative responses

when you first employ these techniques; this is commonly occurs for approximately the first three times that the techniques are used. Try to gain perspective; when negative responses are observed, focus on the deep learning and increased student engagement and remember that students may be resisting your attempts to cause them to work hard.

There are several important factors to consider regarding groups. Keep group sizes small; three persons is ideal. If you need to employ larger sized groups, begin with an initial part of the exercise that includes three people and then combine the two small groups into a group of six for the rest of the exercise. Vary the composition of groups. This helps to increase learning by exposing individuals to many different perspectives and by making the class activities more interesting and exciting. Carefully set up each activity. For example, in an activity that considers conflict management styles, groups could be asked to summarize each style and to tell how it applies to a specific scenario that is given by the instructor. Allow adequate time for groups to perform activities. This requires observing each group, monitoring progress, and asking groups how much time they need to finish; completing activities before all groups are finished is frustrating to students who are deeply engaged with course material. Provide adequate summary time at the end of an activity. This can be accomplished by having students write a minute paper about the group feedback they received or about what they learned in the activity; this can be followed by the instructor enumerating the points that the students should have learned and asking the students if they would like more information about any of the points. An alternate summary method is to ask one or two groups to report to the class about what they learned from each other and about how they answered the questions, addressed the issues or covered the points.

6. Acknowledgements

This work was supported in part by a grant from the Undergraduate Learning Pool of the Ohio University 1804 Endowment.

Many of the techniques described in this paper were evolved from ideas learned from from workshops conducted by Ohio University's Center for Teaching Excellence and Center for Writing Excellence and from *Engaging Ideas*: *The Professor's Guide to Integrating Writing, Critical Thinking, and Active Learning in the Classroom* [10].

7. References

1. I. JACOBSON, G. BOOCH and J. RUMBAUGH, *The Unified Software Development Process*, 1998, Addison Wesley Longman, Inc.

2. R. F. VERDERBER and K.S. VERDERBER, *Communicate!*, 10th Edition, 2002, Wadsworth Publishing Company.

3. E. ARONSON et al., *The Jigsaw Classroom*, Sage Publications, Beverly Hills, CA, 1978.

4. D. W. JOHNSON et al., *Active Learning: Cooperative in the College Classroom*, Interaction Book Company, Edina, MN, 1991.

5. B. J. MILLIS and P. G. COTTELL, Jr., *Cooperative Learning for Higher Education Faculty*, Oryx Press, Phoenix.

6. A. W. CHICKERING and Z. F. GAMSON, "Seven Principles for Good Practice in Undergraduate Education," *AAHE Bulletin*, 3-7, 1987.

7. C. BAZERMAN and D. R. RUSSELL, eds. *Landmark Essays on Writing Across the Curriculum*. Hermagoras Press, Davis, CA, 1994.

8. J. EMIG, "Writing as a Mode of Learning," in *Landmark Essays on Writing Across the Curriculum*, Charles Bazerman and David R. Russell, eds., Hermagoras Press, Davis, CA,1994.

9. C. THAISS, *The Harcourt Brace Guide to Writing Across the Curriculum*, Harcourt Brace, New York, NY, 1998.

10. J. C. BEAN, *Engaging Ideas*: *The Professor's Guide to Integrating Writing, Critical Thinking, and Active Learning in the Classroom*, Jossey-Bass, San Francisco, CA, 1996.

11. W. PERRY, *Forms of Intellectual and Ethical Development in the College Years: A Scheme*, Holt, Rinehart and Winston, New York, NY, 1970.

12. W. PERRY, "Cognitive and Ethical Growth: The Making of Meaning," in *The Modern American College*, Arthur W. Chickering et al., eds., Jossey-Bass, San Francisco, CA, 1981.

13. L. R. WELCH, *CS 456: Software Design and Development*, http://zen.ece.ohiou.edu/cs456.html.